E.T.S.I.S. TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

PLAN 2000

TEMA:	Sistemas Embebidos
TÍTULO:	Técnicas de diseño de pruebas en sistemas embebidos
AUTOR:	Julián Montero Somavilla
TUTOR:	Julio Medina Cano **Vº Bº.**
DEPARTAMENTO:	SEC

Miembros del tribunal calificador:

PRESIDENTE:	Juana María Gutiérrez Arriola
VOCAL:	Julio Medina Cano
SECRETARIO:	Agustín Rodríguez Herrero
Fecha de lectura:	27 de Marzo de 2014

Calificación: **El secretario,**

RESUMEN:

El software es, cada vez más, una parte muy importante de cualquier circuito electrónico moderno, por ejemplo, un circuito realizado con algún tipo de microprocesador debe incorporar un programa de control por pequeño que sea.

Al utilizarse programas informáticos en los circuitos electrónicos modernos, es muy aconsejable, por no decir imprescindible, realizar una serie de pruebas de calidad del diseño realizado.

Estas pruebas son cada vez más complicadas de realizar debido al gran tamaño del software empleado en los sistemas actuales, por este motivo, es necesario estructurar una serie de pruebas con el fin de realizar un sistema de calidad, y en algunos casos, un sistema que no presente ningún peligro para el ser humano o el medio ambiente.

Se realizará un estudio de las técnicas de diseño de pruebas para el software empleado en los circuitos electrónicos.

Agradecimientos

Con la mayor gratitud por los esfuerzos realizados para que yo lograra terminar mi carrera profesional.

A mi madre que es el ser más maravilloso de todo el mundo. Gracias por el apoyo moral, tu cariño y comprensión que desde niño me has brindado, por guiar mi camino y estar junto a mí en los momentos más difíciles.

A mi padre porque desde pequeño ha sido para mí un gran hombre al que siempre he admirado. Gracias por apoyarme con energía durante toda mi vida.

A mi hermana que ha sido un gran ejemplo a seguir, sin ella no sería la persona que soy.

Con cariño, respeto y admiración.

Resumen

El software es, cada vez más, una parte muy importante de cualquier circuito electrónico moderno, por ejemplo, un circuito realizado con algún tipo de microprocesador debe incorporar un programa de control por pequeño que sea.

Al utilizarse programas informáticos en los circuitos electrónicos modernos, es muy aconsejable, por no decir imprescindible, realizar una serie de pruebas de calidad del diseño realizado.

Estas pruebas son cada vez más complicadas de realizar debido al gran tamaño del software empleado en los sistemas actuales, por este motivo, es necesario estructurar una serie de pruebas con el fin de realizar un sistema de calidad, y en algunos casos, un sistema que no presente ningún peligro para el ser humano o el medio ambiente.

Esta propuesta consta de la explicación de las técnicas de diseño de pruebas que existen actualmente (por lo menos las más básicas ya que es un tema muy extenso) para realizar el control de calidad del software que puede contener un sistema embebido.

Además, muchos circuitos electrónicos, debido a su control o exigencia hardware, es imprescindible que sean manipulados por algún programa que requiera más que un simple microprocesador, me refiero a que se deban controlar por medio de un pequeño programa manipulado por un sistema operativo, ya sea Linux, AIX, Unix, Windows, etc., en este caso el control de calidad se debería llevar a cabo con otras técnicas de diseño. También se puede dar el caso que el circuito electrónico a controlar se deba hacer por medio de una página web.

El objetivo es realizar un estudio de las actuales técnicas de diseño de pruebas que están orientadas al desarrollo de sistemas embebidos.

Abstract

Software is increasingly a very important part of any modern electronic circuit, for example, a circuit made with some type of microprocessor must incorporate a control program no matter the small it is.

When computer programs are used in modern electronic circuits, it is quite advisable if not indispensable to perform a series of quality tests of the design.

These tests are becoming more and more difficult to be performed due to the large size of the software used in current systems, which is why it is necessary to structure a series of tests in order to perform a quality system, and in some cases, a system with no danger to humans or to the environment.

This proposal consists of an explanation of the techniques used in the tests (at least the most basic ones since it is a very large topic) for quality control of software which may contain an embedded system.

In addition, a lot of electronic circuits, due to its control or required hardware, it is essential to be manipulated by a program that requires more than a simple microprocessor, I mean that they must be controlled by means of a small program handled by an operating system, being Linux, AIX, Unix, Windows, etc., in this case the quality control should be carried out with other design techniques.

The objective is to study the current test design techniques that are geared to the development of embedded systems. It can also occur that the electronic circuit should be controlled by means of a web page.

Contenido

Tablas

Figuras

Acrónimos

Técnicas de diseño de pruebas en sistemas embebidos

1. INTRODUCCIÓN

1.1. FUNDAMENTOS DE LAS PRUEBAS

1.1.1. Introducción

Las pruebas de cualquier sistema son operaciones técnicas que consisten en la comprobación de una o más características de un sistema, proceso o servicio determinado de acuerdo con un procedimiento específico (Broekman, y otros, 2003).

Aunque existen muchas definiciones del concepto de pruebas, de una manera u otra, todas ellas contienen aspectos similares. Cada una de las definiciones se centra en una parte de un estándar (por ejemplo, el correcto funcionamiento, los requisitos, etc.). Es importante saber exactamente lo que se va a probar (**el objetivo de las pruebas**), sobre qué se va a probar (**la base de pruebas**) y cómo se va a probar (**las técnicas y los métodos de pruebas**).

El principal objetivo de las pruebas es encontrar defectos. Las pruebas tienen como objetivo mostrar la falta de calidad del sistema. Se trata de establecer la diferencia entre el sistema final y los requisitos que debe cumplir el sistema. Dicho de manera positiva, el objetivo es crear un sistema de calidad.

1.1.2. Principios fundamentales de las pruebas

Las pruebas pueden demostrar que existen defectos en el sistema, pero no pueden probar que no los hay. Las pruebas reducen la probabilidad de que queden defectos sin descubrir en el sistema, pero, incluso si no se encuentran defectos, no quiere decir que el sistema esté libre de ellos.

Probar todo (todas las combinaciones de entrada con todas las condiciones iniciales) no es posible salvo en casos triviales. En vez de pruebas exhaustivas, se utilizan los términos de riesgo y prioridades para centrar los esfuerzos en las pruebas que más interesan.

Las actividades de prueba deben comenzar lo antes posible en el ciclo de vida de desarrollo del sistema y debe centrarse en objetivos definidos. Si se realiza de esta forma, la probabilidad de encontrar defectos es mayor que si se deja todo para el final.

Si las mismas pruebas se repiten una y otra vez, con el tiempo, el mismo conjunto de casos de prueba ya no encuentran nuevos errores. Para evitar esto, los casos de prueba deben ser examinados y revisados con regularidad.

1.2. SISTEMAS EMBEBIDOS

1.2.1. Introducción

Sistema embebido es uno de esos términos que en realidad no dice qué es exactamente lo que se trata. Es un término genérico para una amplia gama de sistemas que cubre, por ejemplo, teléfonos móviles, sistemas ferroviarios, audífonos, etc. Sin embargo, todos los sistemas embebidos tienen una característica común que es controlar algún hardware específico.

Un sistema embebido o empotrado se puede definir como un sistema diseñado para realizar una función específica en tiempo real. Estos sistemas realizan funciones de control, procesamiento y/o monitorización.

La Figura 1.1 muestra un diseño genérico que es aplicable prácticamente a todos los sistemas embebidos, señalando los componentes típicos.

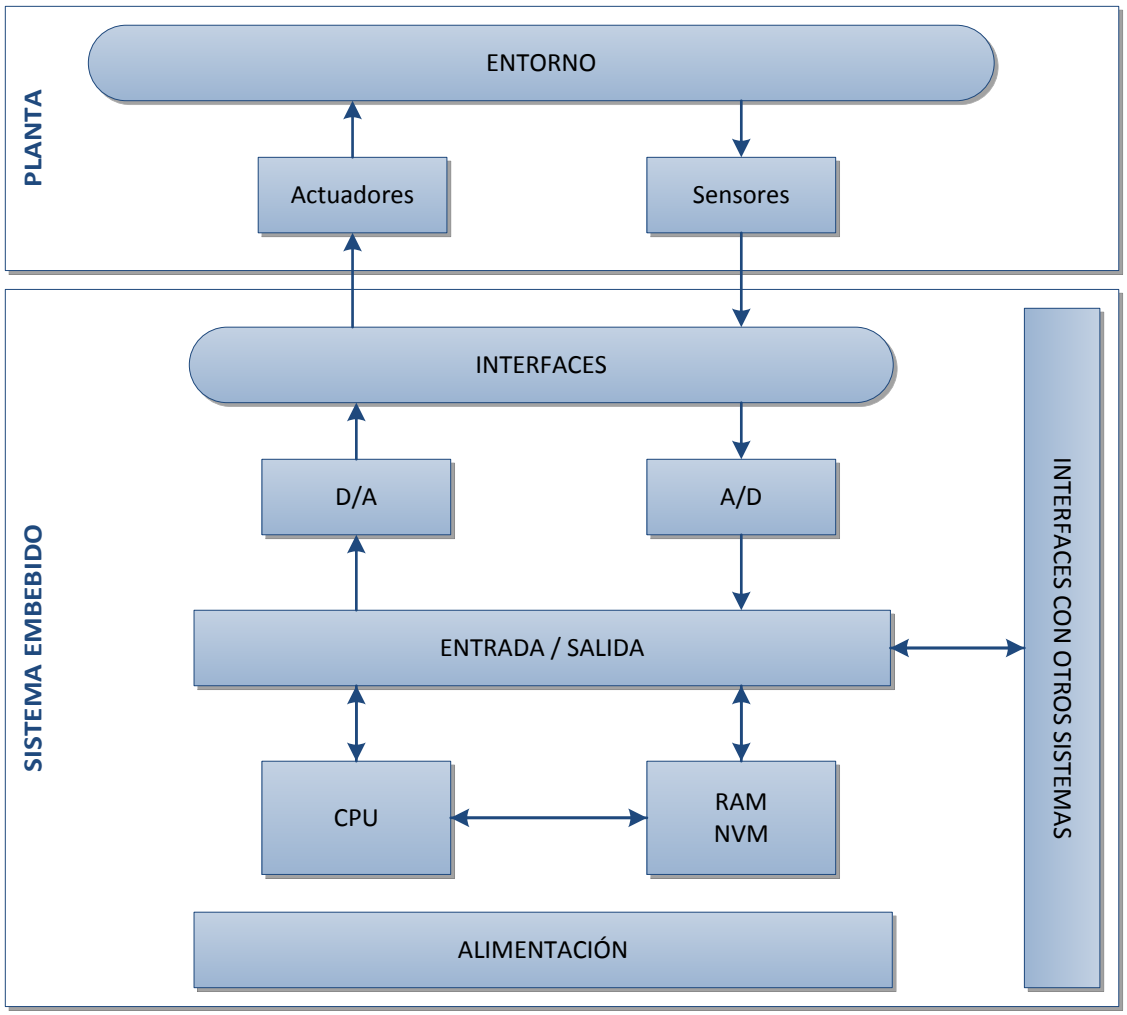


Figura 1.1: Esquema genérico de un sistema embebido

Un sistema embebido interactúa con el mundo real, recibe señales a través de sensores y las envía a través de actuadores, que de alguna manera manipulan el entorno. El entorno junto con los actuadores y los sensores forman lo que se define como la planta.

El software del sistema se almacena en cualquier tipo de memoria no volátil (NVM), esto se conoce comúnmente como memoria de programa. Esta memoria suele ser ROM, pero también puede ser tarjetas de memoria flash o incluso un disco duro o un CD-ROM. El software es compilado en un procesador, la unidad de procesamiento, que normalmente requiere una cierta cantidad de RAM, o memoria de datos, para funcionar. La unidad de procesamiento sólo puede procesar señales digitales, mientras que la planta, posiblemente, funciona con señales analógicas. Para que exista un entendimiento se deben llevar a cabo conversiones A/D y D/A. La unidad de procesamiento maneja todas las señales de entrada y salida (E/S) a través de una capa dedicada. El sistema embebido interactúa con la planta, posiblemente, a través de otras interfaces específicas. Los sistemas embebidos pueden extraer energía de una fuente de alimentación general o pueden poseer su propia fuente de alimentación, tal como las baterías.

Para ilustrar en detalle la definición de sistema embebido o empotrado, se muestra en la Figura 1.2 un sistema relacionado con el funcionamiento de la velocidad de crucero de un vehículo.

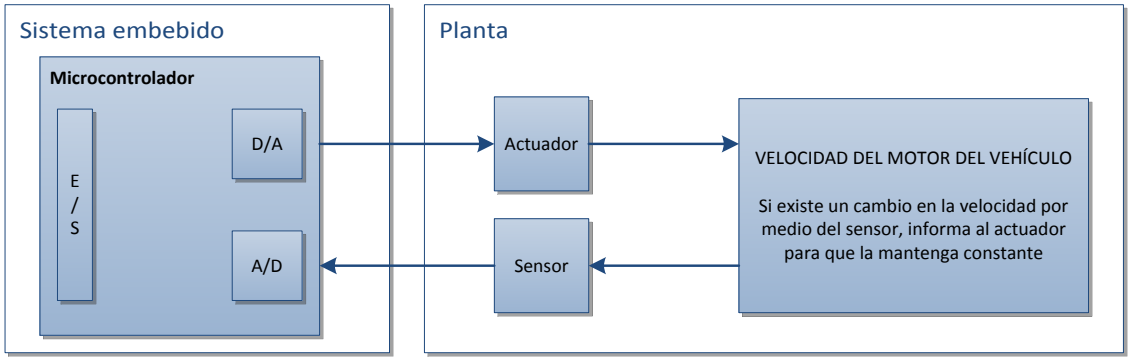


Figura 1.2: Control de crucero de un vehículo

El microcontrolador es el centro del sistema embebido, en él se encuentran todos los periféricos necesarios para controlar la planta. Por medio de los sensores se detectan cambios imprevistos en el funcionamiento del sistema y se actúa inmediatamente sobre el microcontrolador por medio de los denominados actuadores.

Para el acceso a esta información se utilizan conversores A/D y D/A ya que las señales utilizadas por ambos sistemas (sistema embebido y planta) son de naturaleza distinta (digital y analógica).

1.2.2. Análisis

Obviamente, las pruebas que se realizan en los teléfonos móviles son diferentes a las pruebas de un reproductor de video o de un sistema de control de crucero de un coche. Cada uno de ellos necesita medidas concretas en el enfoque de pruebas para cubrir temas específicos de ese sistema. Por lo tanto, no tiene sentido buscar un enfoque de pruebas general para los sistemas embebidos.

Aunque existen muchas razones por lo que los diferentes sistemas embebidos deben ser probados de manera muy diferente, también hay muchos problemas similares que tienen soluciones parecidas y que están involucrados en cualquier enfoque de pruebas. Debe aplicarse algún tipo de principio básico de pruebas a todos los proyectos de pruebas en sistemas embebidos, pero de alguna manera se deben diferenciar las medidas específicas para hacer frente a los problemas de las pruebas de sistemas particulares.

2. ORGANIZACIÓN

2.1. MÉTODO TEMB

2.1.1. Introducción

TEmb (Method for Testing Embedded Software) es un método que ayuda a montar un enfoque de pruebas adecuado para un sistema embebido en particular (Broekman, y otros, 2003).

La base del enfoque de las pruebas para cualquier sistema embebido consta de los elementos genéricos comunes a cualquier planteamiento de pruebas estructurado. Por ejemplo, la planificación de un proyecto según un ciclo de vida determinado, la aplicación de técnicas estandarizadas, los entornos dedicados de las pruebas, informes, etc. Estos elementos están relacionados con el ciclo de vida, la infraestructura, las técnicas y organización.

En las etapas iniciales de un proyecto hay que elegir las medidas que se incluirán en el enfoque de pruebas. En el método TEmb se basa en el análisis de los riesgos y de las características del sistema.

2.1.2. Enfoque de las pruebas

Aunque se tengan sistemas muy diferentes, las pruebas realizadas para todos estos sistemas no tienen que ser completamente distintas. Es muy probable que existan características comunes para algunos sistemas y, gracias a esto, no es necesario realizar un enfoque de pruebas para cada sistema.

Los sistemas que pertenecen al mismo grupo poseen cualidades similares que se pueden abordar de una manera parecida. Debido a que se proporciona un enlace entre las características del sistema y las medidas adecuadas, el montaje de un enfoque de pruebas adecuado se convierte en una tarea fácil.

La ventaja de centrarse en las características del sistema es que se utiliza un conjunto limitado de pruebas para cubrir una enorme variedad de sistemas.

Por otro lado, con el análisis de los riesgos del sistema se elige qué hacer y qué no hacer en el enfoque de las pruebas, es decir, qué probar y que no probar del sistema.

2.1.3. El ciclo de vida

Las principales actividades que se realizan en el ciclo de vida de las pruebas son la fase de planificación y control, la fase de preparación, la fase de especificación, la fase de ejecución y la fase de finalización.

La Figura 2.1 muestra una pequeña descripción gráfica del ciclo de vida de las pruebas.

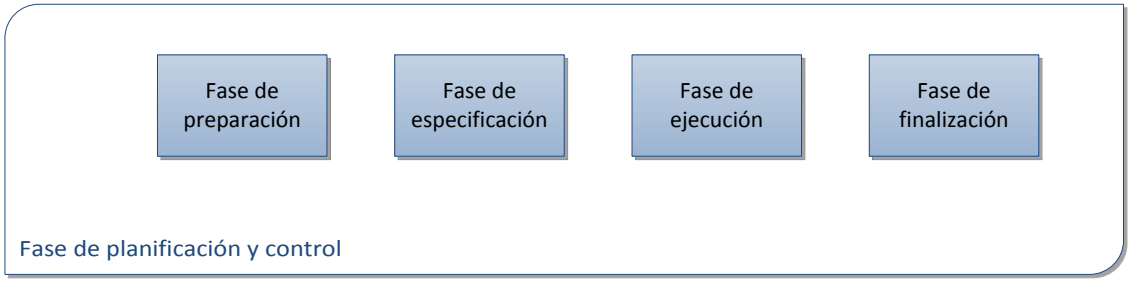


Figura 2.1: Ciclo de vida de las pruebas

La idea básica del ciclo de vida es tratar de hacer el mayor número de actividades tan pronto como sea posible. Sería una pérdida de tiempo si se empieza a analizar qué casos de prueba son similares cuando se está realizando la ejecución de las pruebas, por este motivo, se debe realizar este estudio en una fase previa, la fase de especificación. Sería igualmente molesto si no se puede comenzar a especificar casos de prueba durante la fase de especificación debido a que todavía no se conocen las técnicas de diseño que se van a utilizar y la forma en que se pueden aplicar a las especificaciones del sistema. Esto debería haber sido hecho en la fase anterior, la preparación.

Aunque el ciclo de vida puede mostrar más actividades de las que principalmente se han pensado en un principio, no necesariamente se deben agregar estas actividades adicionales. Se limita a hacer una distinción más clara entre las cosas que hay que hacer y se presentan de manera más eficiente y eficaz. Cuando se aplica con sabiduría, el ciclo de vida es un mecanismo poderoso para ahorrar tiempo.

2.2. MODELO EN V

2.2.1. Introducción

En los sistemas embebidos, el objetivo no es probar sólo el código ejecutable. El sistema, generalmente, se desarrolla en una serie de aspectos que lo hacen más real. En primer lugar, el modelo del sistema se basa en un PC que simula el

comportamiento del sistema. Cuando el modelo se encuentra en un estado correcto, el código se genera a partir del modelo y se realiza un prototipo. El hardware de los prototipos se sustituye gradualmente por el hardware real hasta que finalmente se construye el sistema. La razón de la construcción de esos sistemas de este modo es, por supuesto, porque es más barato y rápido a la hora de cambiar el prototipo por el producto final, e incluso más barato y rápido a la hora cambiar el modelo.

2.2.2. Descripción del modelo

El modelo en V es un modelo de desarrollo que tiene en cuenta el siguiente fenómeno. En principio, cada una de las etapas del producto (modelo, prototipo, producto final) sigue un ciclo de desarrollo en V completo, incluyendo el diseño, construcción y actividades de las pruebas. La esencia es que las diferentes versiones físicas del mismo sistema se desarrollan cada una teniendo la misma funcionalidad requerida al principio. Esto significa, por ejemplo, que la funcionalidad puede ser probada para el modelo, así como para el prototipo y el producto final. Por otra parte, en el modelo no pueden ser probadas determinadas propiedades técnicas y deben ser probadas en el prototipo, por ejemplo, el impacto de las condiciones del entorno.

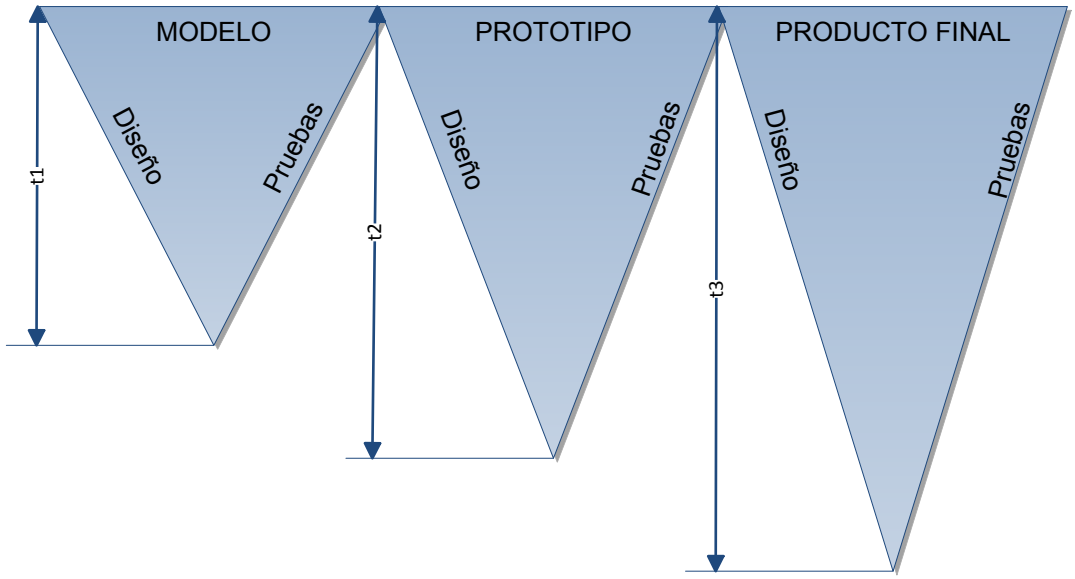


Figura 2.2: Modelo en V

El modelo en V muestra tres formas consecutivas (modelo, prototipos y producto final). Es importante entender que esta es una representación simplificada del proceso de desarrollo para los sistemas embebidos. No debe considerarse como un proceso secuencial sencillo en el que el prototipo está diseñado, construido y probado a la

primera para ser un producto final. El modelo en V es iterativo por naturaleza. En realidad, el desarrollo de un sistema embebido es un proceso complejo.

2.3. PLAN DE PRUEBAS

2.3.1. Introducción

El plan de pruebas proporciona un mecanismo estructurado para controlar el proceso de pruebas. Por ejemplo, si se tiene un sistema con varias partes hardware y cientos de miles de líneas de código software, llevar a cabo todas las pruebas puede ser una labor compleja que implica muchas tareas diferentes y pruebas en varios puntos del sistema. Si se lleva un control de todas estas pruebas es muy posible que la calidad del sistema sea la deseada.

2.3.2. Características de las pruebas

Un sistema puede ser probado desde diferentes puntos de vista: funcional, rendimiento, facilidad de uso, etc. Estas son las características de calidad que describen los diversos aspectos del comportamiento del sistema. Las normas, por ejemplo ISO 9126, definen un conjunto de características que se pueden utilizar para este propósito. En la práctica, se combinan varias en una sola prueba. Esto da lugar al concepto de tipo de prueba.

Un tipo de prueba es un grupo de actividades con el objetivo de probar un sistema con un conjunto de características de calidad relacionadas.

2.3.3. Niveles de las pruebas

Un nivel de una prueba es un conjunto de actividades que se organizan y gestionan como una entidad.

Los diferentes niveles están relacionados con el ciclo de vida del sistema. Las partes del sistema se prueban de forma aislada para comprobar que se ajustan a las especificaciones técnicas. Cuando las distintas partes del sistema son de buena calidad, se integran en una formando un subsistema o el sistema completo. Estas a su vez son probadas para comprobar si se ajustan a los requisitos deseados.

A menudo se hace distinción entre las pruebas de bajo nivel y alto nivel. Las pruebas de bajo nivel están relacionadas con partes aisladas, ejecutadas al inicio del ciclo de vida en un entorno de desarrollo similar. Las pruebas de alto nivel se realizan sobre el sistema o subsistemas, ejecutadas más tarde en el ciclo de vida en un entorno real o simulado.

2.3.4. Plan de pruebas maestro

Si cada nivel define las mejores cosas que hay que hacer, existe un riesgo de que algunas pruebas se realicen más de una vez y otras se omitan.

Para resolver este problema se gestiona un plan de pruebas general que, por supuesto, es mejor para coordinar los diversos niveles: evita redundancias y omisiones, y asegura que se lleva a cabo una estrategia coherente global de las pruebas. Se deben tomar decisiones en cuanto a qué nivel se adecúa más a los requisitos del sistema y a las características de calidad.

En el plan de pruebas general se definen las tareas y los límites de cada nivel. A este plan se le suele denominar “plan de pruebas maestro”.

Un plan de pruebas maestro puede ser visto como la combinación de lo que tiene que ser probado (características de las pruebas) y las actividades que se van a realizar (niveles de las pruebas). El plan de pruebas maestro sirve como una base en cada nivel.

2.3.5. Actividades

Se debe comenzar lo antes posible con los preparativos del proceso de pruebas. Esto se hace por medio del desarrollo de un plan de pruebas maestro. Se inicia con la formulación de los objetivos y las responsabilidades generales sobre el análisis y la definición del alcance del plan de pruebas maestro. A continuación, se realiza un análisis global del sistema y se lleva a cabo el proceso de desarrollo. El siguiente paso es utilizar esta información y exponer qué medidas tomar con respecto a las pruebas para asegurar la calidad del sistema.

3. TIPOS DE PRUEBAS

3.1. INTRODUCCIÓN

3.1.1. Resumen

Existen muchos tipos de pruebas que se pueden realizar al probar un sistema embebido. En este apartado se describen las siguientes:

Tabla 3.1: Tipos de pruebas

TIPOS DE PRUEBAS	
TIPO	DESCRIPCIÓN
Pruebas unitarias	Comprueban el funcionamiento del sistema por separado
Pruebas de integración	Comprueban el funcionamiento de las interfaces entre los componentes o partes del sistema
Pruebas de sistema	Comprueban el comportamiento de todo el sistema
Pruebas funcionales	Comprueban la funcionalidad del sistema, es decir, en lo que hace este sistema
Pruebas de regresión	Comprueban cambios en el software provocados por modificaciones
Pruebas de rendimiento	Comprueban la rapidez de un sistema en realizar ciertas tareas en unas condiciones particulares de trabajo.

3.2. PRUEBAS UNITARIAS

3.2.1. Introducción

Las pruebas unitarias, también conocidas como pruebas de componentes, se utilizan para encontrar defectos y para comprobar el funcionamiento del sistema por separado, por ejemplo, en módulos, programas, objetos, clases, campos, etc.

Una característica importante de las pruebas unitarias es que se pueden realizar aisladas del resto del sistema. Este tipo de pruebas pueden ser funcionales, de rendimiento o estructurales. Los casos de prueba se obtienen del diseño del software o de los modelos de datos.

3.2.2. Enfoque de las pruebas unitarias

Para realizar correctamente las pruebas unitarias es necesario conocer el software del sistema que se está probando. Los defectos detectados se deben solucionar tan pronto como se encuentran, sin necesidad de documentar las incidencias encontradas durante las pruebas.

Un enfoque en las pruebas unitarias o de componentes es preparar y automatizar los casos de prueba antes de la codificación. Esto se denomina prueba de primera aproximación. Este enfoque es altamente iterativo y se basa en el ciclo de vida del desarrollo del sistema. Se debe realizar una construcción e integración de pequeñas partes de código para la posterior ejecución de las pruebas unitarias hasta que todo funcione correctamente en el sistema.

3.3. PRUEBAS DE INTEGRACIÓN

3.3.1. Introducción

Las pruebas de integración se utilizan para comprobar el funcionamiento de las interfaces entre los componentes o partes del sistema. También son usadas en interacciones en diferentes partes de un sistema, tal como un sistema operativo, un sistema de archivos, etc.

3.3.2. Enfoque de las pruebas de integración

Cuanto mayor sea el alcance de la integración, más difícil se hace aislar los fallos de una interfaz específica, lo que puede conducir a un mayor riesgo. Esto lleva a diferentes enfoques para las pruebas de integración.

Uno de estos enfoques es que todos los componentes o sistemas se integran al mismo tiempo, con lo cual, lo que se pone a prueba es el sistema completo. Esto se conoce como pruebas de integración Big-Bang. Las pruebas Big-Bang tienen la ventaja de que todo está terminado antes de comenzar las pruebas. No hay necesidad de simular partes del sistema que están por terminar. La principal desventaja es que es difícil encontrar la causa de los defectos. Por lo tanto, la integración Big-Bang puede parecer una buena idea en la planificación del proyecto si se es optimista y no se espera encontrar ningún problema.

Otro enfoque es que todos los programas son integrados uno por uno y, después de cada paso, se lleva a cabo una prueba (prueba incremental o gradual). El enfoque incremental o gradual tiene la ventaja de que los defectos se detectan a tiempo en un conjunto más pequeño siendo relativamente fácil detectar la causa. Una desventaja es que puede llevar mucho tiempo desarrollar los simuladores de las partes que no están integradas aún en el sistema para usarlos en las pruebas. Dependiendo de la arquitectura del sistema, existen varias posibilidades para probar los programas dentro de una integración gradual. Las principales posibilidades son:

- Integración descendente.
- Integración ascendente.

Con una integración descendente, las pruebas se realizan siguiendo la arquitectura del programa de arriba abajo, por ejemplo, a partir del menú de una interfaz de usuario. Las partes del sistema que aún no se han integrado deben ser simuladas.

Al contrario, con una integración ascendente, las pruebas se realizan siguiendo la arquitectura del programa desde la parte inferior hacia arriba. No se encuentran integradas todas las partes del sistema, al contrario, se comienza por la parte más básica uniendo poco a poco todas las partes que lo componen ascendentemente, así hasta completar el sistema. Como en la integración descendente, si se necesitaran, las partes del sistema que aún no se han integrado deben ser simuladas.

La mejor opción es comenzar la integración con las interfaces que se esperan que vayan a causar más problemas. Esto previene defectos importantes al final de la fase de pruebas de integración. Con el fin de reducir el riesgo de descubrir defectos demasiado tarde, la integración debe ser normalmente incremental en vez de Big-Bang.

En cada etapa, la mejor opción es concentrarse exclusivamente en las pruebas de integración. Por ejemplo, si se están integrando el componente A con el componente B para probar la comunicación entre ambos, no se deben enfocar las pruebas en la funcionalidad de cualquiera de ellos.

3.4. PRUEBAS DE SISTEMA

3.4.1. Introducción

Las pruebas de sistema están relacionadas con el comportamiento de todo el sistema. Puede incluir pruebas basadas en los riesgos y en los requisitos, en procesos de negocio, en casos de uso u otras descripciones del funcionamiento del sistema, las interacciones con el sistema operativo y los recursos del sistema. Las pruebas de sistema suelen ser pruebas finales para comprobar que el sistema cumple con la especificación.

3.4.2. Enfoque de las pruebas de sistema

Las pruebas de sistema deben abarcar requisitos funcionales y no funcionales. Las pruebas no funcionales típicas incluyen el rendimiento y la fiabilidad.

Las pruebas de sistema funcionales se inician mediante el uso de las técnicas basadas en la especificación para la característica del sistema que se quiere probar. Por ejemplo, se puede crear una tabla de decisión para las combinaciones de efectos que se describen en las reglas de negocio. Las técnicas basadas en la estructura también se pueden utilizar para probar el rigor de los elementos de pruebas, por ejemplo, la estructura del menú de la interfaz de usuario del sistema, etc.

3.5. PRUEBAS FUNCIONALES

3.5.1. Introducción

Las pruebas funcionales se basan en la funcionalidad del sistema, es decir, en lo que hace este sistema. Esto se describe en los requisitos, en una especificación funcional o en los casos de uso, dependiendo del componente que se quiere probar.

Las pruebas funcionales consideran el comportamiento del sistema y, a menudo, se las conoce también como pruebas de caja negra.

Las pruebas funcionales, en base a la norma ISO 9126, se centran en la adecuación, la interoperabilidad, la seguridad, la precisión y el cumplimiento.

3.5.2. Enfoque de las pruebas funcionales

Las pruebas funcionales se pueden hacer desde dos perspectivas: basadas en los requisitos o en los procesos de negocio.

Las pruebas basadas en los requisitos utilizan los requisitos funcionales del sistema como base para el diseño de las pruebas. Una buena manera de empezar es utilizar una lista de requisitos como inventario inicial de las pruebas o una lista de elementos para probar y no probar. También hay que dar prioridad a las necesidades basadas en los riesgos (si no se ha realizado ya en la especificación) y utilizarlo para dar prioridad a las pruebas. Esto asegurará que las pruebas más importantes y más críticas se incluyan en la base de pruebas.

Las pruebas basadas en el proceso de negocio utilizan el conocimiento de los procesos de negocio. Los procesos de negocio describen los escenarios involucrados en el uso del negocio del día a día del sistema. Los casos de uso se originan en el desarrollo orientado a objetos, pero hoy en día son populares en muchos ciclos de vida de desarrollo. Los casos de uso son una base muy útil para los casos de prueba desde una perspectiva empresarial.

Las técnicas utilizadas para las pruebas funcionales están basadas en la especificación. Las condiciones y los casos de prueba se obtienen de la funcionalidad del componente o del sistema. Como parte del diseño de las pruebas, se puede desarrollar un modelo como un proceso, un modelo de transición de estado o una especificación sencilla en algún tipo de lenguaje.

3.6. PRUEBAS DE REGRESIÓN

3.6.1. Introducción

Esta categoría es un poco diferente a las demás, porque si se ha hecho un cambio en el software, es posible que se haya cambiado la forma en que funciona y su estructura. Sin embargo, lo que se busca con estas pruebas son defectos relativos a los cambios, a pesar de que pueden incluir todos los otros tipos de prueba.

Cuando una prueba falla y se determina que la causa del fallo es un defecto de software, se corrige el defecto y se crea una nueva versión del software. En este caso, se tienen que ejecutar las pruebas otra vez para confirmar que el defecto se ha solucionado. Esto se conoce como prueba de regresión.

Al hacer las pruebas de regresión es importante asegurarse de que se ejecutan exactamente de la misma manera que la primera vez, usando las mismas entradas de datos y el mismo entorno. Si la prueba es correcta no significa que todo el software funciona correctamente, por ahora se puede afirmar que al menos una parte del software es correcto. La solución de este defecto puede haber introducido otro diferente en otro lugar del software. La forma de detectar estos inesperados efectos secundarios es hacer pruebas de regresión.

3.6.2. Enfoque de las pruebas de regresión

Las pruebas de regresión implican la ejecución de los casos de prueba que se han ejecutado antes.

El propósito de las pruebas de regresión es comprobar que las modificaciones sufridas en el sistema o en el entorno no han causado efectos secundarios adversos no deseados y que el sistema todavía cumple con los requisitos.

Las pruebas de regresión se ejecutan cada vez que existen cambios en el software, ya sea como consecuencia de correcciones o por la introducción de una nueva funcionalidad o una modificación.

Las pruebas de regresión, debido al gran volumen de pruebas que pueden contener, se suelen automatizar para poder ejecutarlas todas sin necesidad de perder el tiempo. Puede ser posible y deseable eliminar algunos de los casos de prueba si son repetitivos (pruebas que ejercen las mismas condiciones), o se pueden combinar (si siempre se ejecutan a la vez). Otro enfoque es el de eliminar los casos de prueba que no han encontrado un defecto durante mucho tiempo (aunque este enfoque se debe utilizar con cuidado).

3.7. PRUEBAS DE RENDIMIENTO

3.7.1. Introducción

Las pruebas de rendimiento son pruebas que se realizan para determinar la rapidez de un sistema en realizar ciertas tareas en unas condiciones particulares de trabajo. También pueden servir para validar y comprobar otros atributos de la calidad del sistema, tales como la escalabilidad, fiabilidad y uso de los recursos.

Las pruebas de rendimiento pueden servir para diferentes propósitos: pueden demostrar que el sistema cumple los criterios de rendimiento, pueden comparar dos sistemas para encontrar cuál de ellos funciona mejor o pueden medir que partes del sistema o de carga de trabajo provocan que el conjunto funcione mal.

Es fundamental para alcanzar un buen nivel de rendimiento de un nuevo sistema que los esfuerzos en estas pruebas comiencen en el inicio del proyecto de desarrollo y se amplíen durante su construcción. Cuanto más se tarde en detectar un defecto de rendimiento, mayor es el coste de la solución. Esto es cierto en el caso de las pruebas funcionales, pero mucho más en las pruebas de rendimiento, debido a que su ámbito de aplicación es de principio a fin.

En las pruebas de rendimiento, a menudo es crucial (y con frecuencia difícil de conseguir) que las condiciones de prueba sean similares a las esperadas en el uso real. Esto es, sin embargo, casi imposible en la práctica. La razón es que los sistemas en producción tienen un carácter aleatorio de la carga de trabajo y aunque en las pruebas se intente dar lo mejor de sí para imitar el volumen de trabajo que pueda tener el entorno de producción, es imposible reproducir exactamente la variabilidad de ese trabajo, salvo en el sistema más simple.

3.7.2. Enfoque de las pruebas de rendimiento

Es fundamental detallar las especificaciones de rendimiento (requisitos) y documentarlas en algún plan de pruebas de rendimiento. Idealmente, esto se hace durante la fase de requisitos del desarrollo de cualquier proyecto de desarrollo de sistemas, antes de cualquier esfuerzo de diseño.

Sin embargo, con frecuencia las pruebas de rendimiento no se realizan teniendo en cuenta alguna especificación, es decir, nadie ha expresado cuál es el tiempo máximo de respuesta aceptable para un número determinado de usuarios.

La idea es identificar la parte del sistema más débil. Normalmente existe una parte del sistema que, si responde con mayor rapidez, se traducirá en un funcionamiento del sistema global más rápido.

A veces es una difícil tarea determinar qué parte del sistema representa esta ruta crítica, y algunas herramientas de prueba incluyen (o puede tener añadidos que lo proporcionan) instrumentos que se ejecuta en el servidor (agentes) y que informan de tiempos de transacción, número de accesos a bases de datos, sobrecarga de la red, y

otros monitores del servidor, que pueden ser analizados junto con los datos principales de las estadísticas de rendimiento.

4. COBERTURA

4.1. TIPO Y RADIO DE COBERTURA

4.1.1. Introducción

La cobertura es la relación entre lo que puede ser probado y lo que se quiere o pretende poner a prueba. En otras palabras, la cobertura es la cantidad de todas las posibilidades que se pueden probar y son realmente probadas. Esto significa que hay dos fenómenos que en conjunto determinan la cobertura:

- Tipo de cobertura.
- Radio de cobertura

4.1.2. Tipo de cobertura

El tipo de cobertura es la forma en que la cobertura de las situaciones de prueba se deduce de la base de pruebas. Indica qué tipo de posibilidades están involucradas. Estas se elaboran posteriormente en las situaciones de prueba.

Por ejemplo, esto es posible mirando las combinaciones posibles de las trayectorias en un algoritmo. Pero en ese mismo algoritmo, también se puede ver las posibilidades de cubrir un resultado u otro en cada punto de decisión. Estas son dos formas diferentes de cobertura.

4.1.3. Radio de cobertura

El radio de cobertura es el porcentaje de situaciones de prueba, tal como se define por el tipo de cobertura, que está cubierto por las pruebas.

Indica cuáles de las posibilidades anteriores se van a probar. Esto se denomina relación de cobertura y se expresa generalmente como un porcentaje. Se muestra qué parte de todas las alternativas han sido realmente probadas. El número (porcentaje) no tiene sentido hasta que quede claro que se trata de posibilidades, es decir, cuando se asocia con el tipo de cobertura.

4.2. APLICACIÓN DE LA COBERTURA

4.2.1. Importancia de la cobertura

Dado que es imposible probar todo y por lo tanto nos vemos obligados a probar sólo un pequeño conjunto de todas las posibilidades, ¿cuáles sería el mejor conjunto de posibilidades?

La cobertura tiene que ver con el deseo de encontrar el mayor número de defectos posibles con el menor número de casos de prueba. En lugar de simplemente probar cualquier conjunto de opciones, el objetivo es elaborar un conjunto de casos de prueba para conseguir que se encuentren todos los defectos que están o pudieran estar presentes en el sistema. Nunca se puede estar seguro de que todos los defectos se han encontrado o que se ha encontrado, por ejemplo, el 60% de todos los defectos. Después de todo, no se sabe cuántos defectos hay en realidad. Lo que se puede demostrar es la cobertura realizada por la prueba. Y esto otorga un cierto nivel de confianza de que la probabilidad de cualquier defecto que queda en el sistema es pequeña. En resumen, cuanto mayor sea la cobertura, menor es la probabilidad de que cualquier defecto permanezca en el sistema.

La decisión de probar más a fondo se traduce a una decisión para lograr una cobertura más amplia. En principio, existen tres opciones:

- Un tipo de cobertura más completo.
- Múltiples tipos de cobertura.
- Una tasa de cobertura más alta de un tipo de cobertura específico.

4.2.2. Enfoque de la cobertura

Debe quedar claro a estas alturas que una afirmación como “se quiere probar con un radio de cobertura del 75%” no dice mucho por las siguientes razones:

- ¿De qué tipo de cobertura se está hablando?
- ¿Por qué el 25% restante no necesitan ser cubierto?, en otras palabras, ¿por qué no simplemente cubrir el 100%?

No se puede hablar de la cobertura en sí, pero si hablar de muchas formas de cobertura (es decir, los tipos de cobertura). La elección más completa para las pruebas deber ser elaborada principalmente con la elección del tipo de cobertura. El uso de un

tipo de cobertura implica que se tiene como objetivo una relación de cobertura del 100%.

En general, los diferentes tipos de cobertura no se pueden comparar en términos de “X es mejor que Y”. Los diferentes tipos de cobertura tienden a complementar y no sustituir a la otra.

5. TÉCNICAS DE DISEÑO

5.1. CONCEPTOS GENERALES

5.1.1. Introducción

En el caso ideal, las pruebas dan la seguridad de que el sistema se comporta como se desea en cualquier circunstancia. En realidad, no todas las circunstancias pueden ser probadas, sólo un pequeño conjunto de funcionalidades del sistema definido en la parte de diseño puede ser probado (Koomen, y otros, 2006).

El diseño conduce a una estructura jerárquica de situaciones, casos y guiones de prueba.

- Cada situación de prueba se produce al menos en un caso de prueba.
- Un caso de prueba lógico cubre una o más situaciones de prueba.
- Un caso de prueba lógico es convertido en un caso de prueba físico.
- Cada caso de prueba físico se presenta en un guion de pruebas.

5.1.2. Situaciones de prueba

Una situación de prueba es una condición aislada en las que el objetivo muestra un comportamiento específico que tiene que ser probado.

La cobertura específica y el procedimiento de construcción de la prueba elegida para lograrlo determinan exactamente qué casos son definidos.

5.1.3. Casos de prueba

Un caso de prueba se utiliza para comprobar si el sistema muestra el comportamiento deseado en determinadas circunstancias.

Por lo tanto, un caso de prueba debe contener todos los ingredientes para determinar el comportamiento del sistema y comprobar si es o no correcto.

Se deben describir los siguientes elementos en todos los casos de prueba, independientemente de la técnica de diseño utilizada:

- Situación inicial.
- Acciones.
- Resultado esperado.

Al contrario de una situación de prueba que se ocupa de un aspecto aislado, un caso de prueba es una unidad completa que se puede ejecutar como una prueba separada.

En el diseño de casos de prueba, primero se crean casos de prueba lógicos que luego se convierten en casos de prueba físicos. Un caso de prueba lógico describe las circunstancias en las que el comportamiento del sistema se examina al indicar que las circunstancias de prueba están cubiertas por el caso. Un caso de prueba físico es la elaboración concreta de un caso de prueba lógico, con opciones de haber sido hechas por los valores de todas las situaciones de entrada y los ajustes de los factores ambientales.

5.1.4. Guion de pruebas

En un guion de pruebas, se combinan múltiples casos de prueba físicos para ejecutarse de una manera eficiente y sencilla.

No existen reglas estrictas que digan que los casos de prueba deban ser combinados en un guion o cuántos se permiten. Se tiene plena libertad siempre y cuando los guiones contengan todos los casos de prueba. Las razones para combinar casos de prueba específicos en un guion son de carácter puramente práctico. Por ejemplo:

- Usan la misma situación inicial.
- Las mismas acciones que requieren mucho tiempo se deben ejecutar para todos los casos de prueba.
- Todos ellos tienen que ver con la misma situación de prueba. Por ejemplo, la secuencia de comandos contiene todos los casos de prueba sobre los clientes en el extranjero.
- Hay casos de prueba dependientes de otros. Antes de la ejecución de algunos casos de prueba, otros deben haber terminado primero.

5.2. ESTRUCTURA DE UNA TÉCNICA DE DISEÑO

5.2.1. Introducción

Una prueba estructurada significa que se han pensado bien los casos de prueba necesarios para alcanzar las metas requeridas, y que se han preparado antes de la ejecución de las pruebas reales. En otras palabras, la prueba estructurada implica que

se apliquen las técnicas de diseño. Una técnica de diseño es un método estandarizado que consiste en realizar casos de prueba a partir de una información de referencia.

5.2.2. Descripción de los pasos a seguir

La mayoría de las técnicas de diseño de pruebas en un proceso estructurado sigue un plan genérico que consiste en pasos discretos:

- Identificar las situaciones de prueba.
- Especificar los casos de prueba lógicos.
- Especificar los casos de prueba físicos.
- Establecer la situación inicial.
- Montar el guion de pruebas.
- Describir los escenarios de pruebas.

Cada técnica de diseño de pruebas tiene como objetivo encontrar ciertos tipos de defectos y alcanzar un cierto tipo de cobertura. Esto tiene que estar relacionado con la información que existe en la base de pruebas, que puede consistir en requisitos funcionales, diagramas de transición de estados, casos de uso o en cualquier otra documentación que especifique el comportamiento del sistema. El primer paso en el proceso del diseño es analizar la base de pruebas e identificar las situaciones que se van a probar. Una técnica de diseño de pruebas puede indicar, por ejemplo, que las situaciones que se evalúan son todas aquellas que se forman con todas las condiciones, tanto verdaderas y falsas, o que cada parámetro de entrada se prueba con los valores válidos y no válidos.

Las situaciones de prueba se transforman en casos de prueba lógicos, que pasan a través del objetivo de las pruebas de principio a fin. Es posible que los casos de prueba lógicos sean los mismos que las situaciones de prueba, pero en general, contienen varias situaciones. Los casos de prueba se denominan “lógicos” porque describen el tipo de situación que se evalúa sin dar valores concretos de los parámetros involucrados. Los casos de prueba lógicos sirven para demostrar que se logran los objetivos previstos y la cobertura establecida.

El siguiente paso es la creación de los casos de prueba físicos. Para cada caso de prueba lógico se eligen valores concretos de entrada y se define la salida resultante en términos concretos. La elección de los valores de entrada puede ser limitada por acuerdos sobre, por ejemplo, el uso de valores de límite o el uso de ciertos datos

iniciales. En general, un caso de prueba físico proporciona toda la información necesaria para ejecutar ese caso de prueba, incluidos los valores de entrada, las medidas de prueba a ejecutar y los resultados esperados.

Con el fin de ejecutar los casos de prueba físicos se debe definir una situación inicial. Esto significa que debe ser instalado un determinado conjunto de datos o que el sistema debe ser puesto en un determinado estado. A menudo, varios casos de prueba físicos pueden comenzar desde la misma situación inicial. Entonces es útil preparar conjuntos de datos iniciales y procedimientos para poner al sistema en un estado determinado, que puede ser reutilizado para muchos casos de prueba.

El paso final es definir la secuencia de comandos a seguir en las pruebas, también llamado guion de pruebas, que coloca las acciones y comprueba el resultado de los casos de prueba individuales en una secuencia óptima para la ejecución de las pruebas. El guion de pruebas sirve como una guía paso a paso en la forma de ejecutar las pruebas. Los casos de prueba físicos en relación con las situaciones iniciales preparadas constituyen la base de este guion.

Como paso opcional, se pueden describir los escenarios donde se van a ejecutar las pruebas. Esto se recomienda en situaciones complejas en las algunos guiones de pruebas tienen dependencias con otros (por ejemplo, el guion B sólo se puede ejecutar después de la finalización exitosa del guion A). Un escenario de pruebas puede ser visto como una especie de “pequeña planificación de pruebas”, que describe el orden en que los guiones de pruebas deben ser ejecutados, qué acciones iniciales son necesarias y las posibles alternativas en el caso de que las cosas vayan mal.

5.2.3. Ventajas

La aplicación de técnicas de diseño de pruebas y la documentación de los resultados obtenidos tienen muchas ventajas para el proceso de pruebas. En definitiva, se incrementa la calidad y el control del proceso de pruebas.

El uso de técnicas de diseño de pruebas da una idea de la calidad y la cobertura de las pruebas, sobre una base de implantación sólida de la estrategia de pruebas, lo que da la cobertura correcta en el lugar correcto.

Como una técnica de diseño de pruebas tiene como objetivo encontrar ciertos tipos de defectos (por ejemplo, en las interfaces, en las validaciones de entrada, o en el

procesamiento), tales defectos serán detectados con mayor eficacia que si se realizan pruebas al azar.

Las pruebas se pueden reproducir fácilmente debido a que el orden y el contenido de la ejecución se han especificado en detalle.

El procedimiento de trabajo normalizado hace que el proceso sea independiente de la persona que establece y ejecuta los casos de prueba. Además, hace de los diseños transferibles y fáciles de mantener.

El proceso de pruebas se puede planificar y controlar más fácilmente debido a la especificación y a que los procesos de ejecución se han dividido en bloques bien definidos.

5.2.4. Pruebas de caja negra y caja blanca

Las técnicas de caja negra se basan en el comportamiento funcional de los sistemas, sin conocimiento explícito de los detalles de implementación. En las pruebas de caja negra el sistema se somete a una entrada, y la salida resultante se analiza para comprobar que se cumple con el comportamiento esperado del sistema.

Las técnicas de caja blanca se basan en el conocimiento de la estructura interna del sistema, es decir, en el código, las descripciones de los programas y el diseño técnico.

5.3. TÉCNICAS DE DISEÑO BÁSICAS

5.3.1. Procesamiento de la lógica

Un principio importante es basar los casos de prueba en la lógica del procesamiento del programa, función o sistema que va a ser probado. Aquí, el procesamiento es visto como un conjunto compuesto de puntos de decisión y acciones. Un punto de decisión consta de una o más condiciones y establece que “SI esta condición ocurre ENTONCES continuar con la acción 1 SINO saltar a la acción 2”. Las diversas combinaciones de decisiones y acciones que pueden ser tomadas consecutivamente se conocen como caminos.

A continuación se muestra un ejemplo de la descripción de la lógica de procesamiento. En él se describe cuando un ascensor debe subir, bajar o parar. Esto depende de su

posición actual, la planta seleccionada por personas en el interior del ascenso, y la planta (fuera del ascensor) donde se solicita que el ascensor se mueva.

Tabla 5.1: Control lógico de un ascensor

SITUACIONES DE PRUEBA	
ESPECIFICACIÓN	EXPLICACIÓN
IF seleccionar planta == “on” THEN	Decisión B1, condición C1
IF destino > actual THEN	Decisión B2, condición C2
ascensor sube	Acción A1
ELSE	
ascensor baja	Acción A2
IF destino = actual OR destino = solicitada THEN	Decisión B3, condición C3, C4
ascensor para	Acción A3

En la situación descrita en la Tabla 5.1, cuando una persona está en la planta baja y pulsa el botón para el tercer piso, y otra persona está en el segundo piso pulsa el botón del ascensor, las acciones posteriores seguirán el camino “B1/B2/A1/B3/A3”. Esto se traduce en que el ascensor sube al segundo piso y luego se detiene.

La lógica de procesamiento puede ser considerada en diferentes niveles. En las pruebas de bajo nivel, la atención se centra en la estructura interna de los programas. Las declaraciones en el programa constituyen entonces las decisiones y acciones. Para las pruebas de alto nivel, los requisitos funcionales pueden ser vistos como lógica de procesamiento.

5.3.2. Clases de equivalencia

Usando este principio se obtienen los casos de prueba. El dominio de entrada (todos los posibles valores de entrada) se divide en “clases de equivalencia”. Esto significa que para todos los valores de entrada en una clase de equivalencia particular, el sistema muestra el mismo tipo de comportamiento (realiza el mismo procesamiento).

Las clases de equivalencia se pueden distinguir entre válidas y no válidas. Los valores de entrada de un resultado no válido, en algún tipo de manejo de excepciones, son tales como la generación de un mensaje de error. Los valores de entrada válidos de una clase de equivalencia deben ser procesados correctamente.

La idea de este principio es que todas las entradas de la misma clase de equivalencia tienen las mismas posibilidades de encontrar un defecto, y que las pruebas con más entradas de la misma clase apenas aumentan la probabilidad de encontrar defectos. En lugar de probar cada posible valor de entrada, es suficiente elegir una entrada de cada clase de equivalencia. Esto reduce considerablemente el número de casos de prueba sin dejar de lograr una buena cobertura.

Esto se muestra en el siguiente ejemplo, donde se somete el comportamiento del sistema a la siguiente condición con respecto a la temperatura:

$$15 \leq Temperatura \leq 40$$

El número de posibles valores para la temperatura es enorme (de hecho, es infinito). Sin embargo, este dominio de entrada puede dividirse en tres clases de equivalencia:

- La temperatura es inferior a 15;
- La temperatura tiene un valor en el rango de 15 a 40;
- La temperatura es superior a 40.

Estos tres casos de prueba son suficientes para cubrir la clase de equivalencia. Por ejemplo, podrían ser elegidos los siguientes valores de temperatura: 10 (no válido), 35 (válido) y 70 (no válido).

La partición de equivalencia también se puede aplicar al dominio de salida del sistema. Los casos de prueba se obtienen para cubrir todas las clases de salida equivalentes.

5.3.3. Análisis del valor límite

Una aplicación importante de este principio es el análisis del valor límite. Los valores que separan las clases de equivalencia se conocen como valores límite.

La idea de este principio es, que los defectos pueden ser causados por fallos “simples” de programación relacionados con el uso erróneo de los límites. Si el programador ha codificado “menor que” cuando debería haber sido “menor o igual que” existe un error en la codificación. Cuando se obtienen los casos de prueba se eligen los valores de estos límites de manera que cada límite se pruebe con un mínimo de dos casos de prueba, uno cuando el valor de entrada es igual al límite y otro una unidad por debajo o por encima de este límite.

Utilizando el ejemplo anterior y suponiendo una tolerancia de 0,1 en los valores de temperatura, los valores límite para ser seleccionados son 14,9 (no válido), 15 (válida), 40 (válida) y 40,1 (no válido).

Un uso más completo de análisis del valor límite es que se eligen tres valores en lugar de dos en cada límite. El valor adicional se elige sólo dentro de la partición de equivalencia definida por el valor límite.

En el ejemplo anterior, se deben probar los dos valores adicionales 15.1 (válido) y 39.9 (válido). Si el programador codifica incorrectamente "*15 == Temperatura*" en lugar de "*15 <= Temperatura*", esto no sería detectado con sólo dos valores límite, pero sería detectado con el valor adicional de 15,1.

El análisis del valor límite y las clases de equivalencia se pueden aplicar al dominio de salida. Supongamos que un mensaje en una pantalla LCD tiene un máximo de cinco líneas. Entonces esto se prueba mediante el envío de un mensaje con cinco líneas (todas las líneas en una pantalla) y uno con seis líneas (la primera línea de la segunda pantalla).

Aplicando el análisis del valor límite se generan más casos de prueba, pero aumentan las posibilidades de encontrar más defectos que con una selección al azar de las clases de equivalencia.

5.3.4. Técnica de diseño formal e informal

Una técnica de diseño formal tiene reglas estrictas sobre cómo se obtienen los casos de prueba. La ventaja es que minimiza el riesgo de que se olvide algo importante. La desventaja del diseño formal es que los casos de prueba sólo pueden ser tan buenos como la especificación en la que se basan. Una especificación pobre del sistema conducirá a una prueba pobre.

Una técnica de diseño de pruebas informal da reglas generales y deja más libertad a la hora de probar. Esta pone más énfasis en la "creatividad" y la "corazonada" acerca de las posibles deficiencias del sistema. Las técnicas de diseño de pruebas informales son menos dependientes de la calidad de la base de pruebas, pero tienen la desventaja de proporcionar poca información acerca del grado de cobertura en relación con la base de pruebas.

5.4. ÁREA DE APLICACIÓN

5.4.1. Introducción

Algunas técnicas de diseño de pruebas son particularmente adecuadas para probar el procesamiento detallado de un componente, mientras que otras son más adecuadas para probar la integración entre las funciones y los datos. Sin embargo, otro grupo tiene la intención de probar la interacción entre los sistemas y el mundo exterior (los usuarios u otros sistemas). La habilidad de las diversas técnicas se relaciona con el tipo de defectos que se pueden encontrar con su ayuda, tales como actividades de validación de entrada incorrecta, procesamiento incorrecto o defectos de integración.

5.4.2. Características de la calidad

Un conjunto de casos de prueba que cubren la funcionalidad a probar en un sistema puede ser inadecuado para probar su rendimiento o fiabilidad. En general, la elección de una técnica de diseño de pruebas depende en gran medida de las características de calidad.

5.4.3. Base de pruebas

Debido a que una técnica de diseño de pruebas es, por definición, una forma estándar de obtener casos de prueba a partir de una base de pruebas, se necesita un tipo específico de base de pruebas. Por ejemplo, el uso de la técnica de “transición de estados” necesita que un modelo basado en el estado del sistema esté disponible. En general, las técnicas de diseño de pruebas formales son más dependientes de la disponibilidad de una base de pruebas que las técnicas de diseño de pruebas informales.

5.5. TÉCNICA DE DISEÑO: TRANSICIÓN DE ESTADOS

5.5.1. Introducción

Muchos sistemas embebidos, o partes de estos sistemas, muestran su comportamiento basándose en estados. El diseño de estos sistemas está basado en el uso de un modelo. La composición de estos modelos sirve como base para el diseño de las pruebas de este tipo de sistemas.

La finalidad de la técnica de diseño de pruebas basada en el estado es verificar las relaciones entre eventos, acciones, actividades, estados y transiciones. Mediante el uso de esta técnica, se puede concluir si el comportamiento de un sistema basado en el estado cumple con las especificaciones establecidas para este sistema. El comportamiento de un sistema se puede clasificar en los siguientes tipos:

- Simple.
- Continuo.
- Basado en estados.

Cuando el comportamiento es simple el sistema siempre responde del mismo modo a una determinada entrada, independientemente de los estados anteriores. Si el comportamiento es continuo el estado actual del sistema depende de estados anteriores de tal manera que no es posible identificar un estado por separado. Sin embargo, si se trata de un comportamiento basado en estados el estado actual del sistema depende de estados anteriores y estos estados se pueden distinguir claramente de otros estados.

5.5.2. Categoría de los defectos

Un comportamiento incorrecto de un sistema que está basado en estados puede tener tres causas.

La primera es que el diagrama de estados no representa una traducción correcta de las especificaciones funcionales del sistema. La técnica de pruebas diseñada no es capaz de revelar estos tipos de defectos debido a que los mismos diagramas se utilizan como la base de pruebas.

Figura 5.1: Traducción incorrecta de las especificaciones

La segunda causa es que los diagramas de estados son sintácticamente incorrectos o inconsistentes. Estos defectos pueden ser revelados por las pruebas estáticas. Si los

defectos encontrados de esta forma se corrigen, entonces el diagrama de estados constituye la base de pruebas dinámicas del sistema.

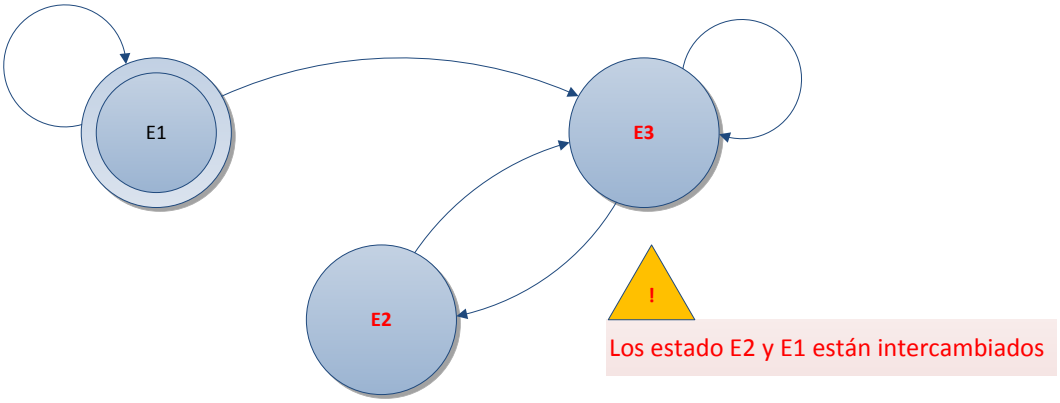
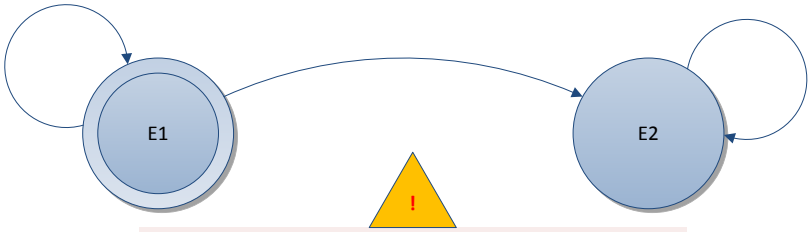


Figura 5.2: Diagrama de estados sintácticamente incorrecto

La tercera causa es una mala traducción de los diagramas de estados en código. Cada vez es más común que esta traducción se realice automáticamente. Si es así, el código generado es una representación exacta del comportamiento de los diagramas de estados. El uso de los diagramas de estados como la base para el diseño de las pruebas en este caso es, por lo tanto, poco útil y la aplicación de la técnica de pruebas basada en el estado es redundante. Sin embargo, si la codificación se realiza sin el uso de herramientas, a continuación, se debe utilizar la técnica de pruebas basada en el estado.



Fragmento de código:

```
…
siwtch(estado)
{
    case 1:
        …
    case 3: // sólo existen dos estados
        ...
}
…
```

Figura 5.3: Traducción incorrecta del diagrama de estados en código

Los defectos más comunes de un software diseñado por medio de diagramas de estado se encuentran en los estados, las acciones, las transiciones y los eventos.

Los defectos relacionados con los estados pueden ser varios, por ejemplo, que existan estados sin transiciones de entrada, lo que provoca que sea inaccesible. Otro defecto podría ser que no existe un estado inicial, sin este estado no se puede predecir dónde termina la transición. El número de estados es importante, si el sistema cuenta con más o menos estados de los que necesita se produciría un error en la ejecución ya que el sistema no cumple con las especificaciones. Un defecto grave puede ser que exista un estado incorrecto, lo que hace que al realizar una transición se llegue a un estado equivocado y como consecuencia pueda afectar gravemente al sistema llegando a la caída del mismo.

Cuando los defectos son provocados por las acciones se tienen algunas situaciones que pueden provocar que el sistema funcione incorrectamente, por ejemplo, si una acción apunta a un estado en lugar de a una transición. Al definir las acciones se debe tener especial cuidado para que el sistema no entre en un bucle infinito. Si se aplican incorrectamente las acciones, el comportamiento del sistema puede que no sea el deseado.

Las transiciones deben tener un estado resultante. Por otro lado, se debe poner especial atención para que no existan transiciones incorrectas o que falte alguna de ellas, el estado resultante puede ser incorrecto.

Los eventos son una parte importante en el diseño de estos sistemas. Si falta uno de ellos, el sistema no funcionaría de la forma esperada. Además, si existe un camino oculto, se lleva a cabo una reacción a un evento definido a pesar de que no esté definida en el diagrama de estados.

5.5.3. Procedimiento

Cuando se prueba el comportamiento basado en el estado, el sistema debe ser comprobado de modo que en respuesta a los eventos de entrada se tomen las acciones correctas y el sistema alcance el estado correcto.

La técnica de pruebas de transición de estados (STT) se aplica al nivel de profundidad de pruebas 1 y utiliza diagramas de estado planos. STT consta de los siguientes pasos:

- Realizar la tabla de estados y eventos.
- Realizar el árbol de transiciones.
- Realizar el guion de los casos de prueba legales.

- Realizar el guion de los casos de prueba ilegales.

5.5.3.1. Tabla de eventos y estados

El diagrama de estados es el punto de partida para la composición de una tabla de estados y eventos, esto representa los estados frente a los eventos. Por ejemplo, el diagrama de la figura 5.1 muestra los estados y los eventos del funcionamiento de un ascensor.

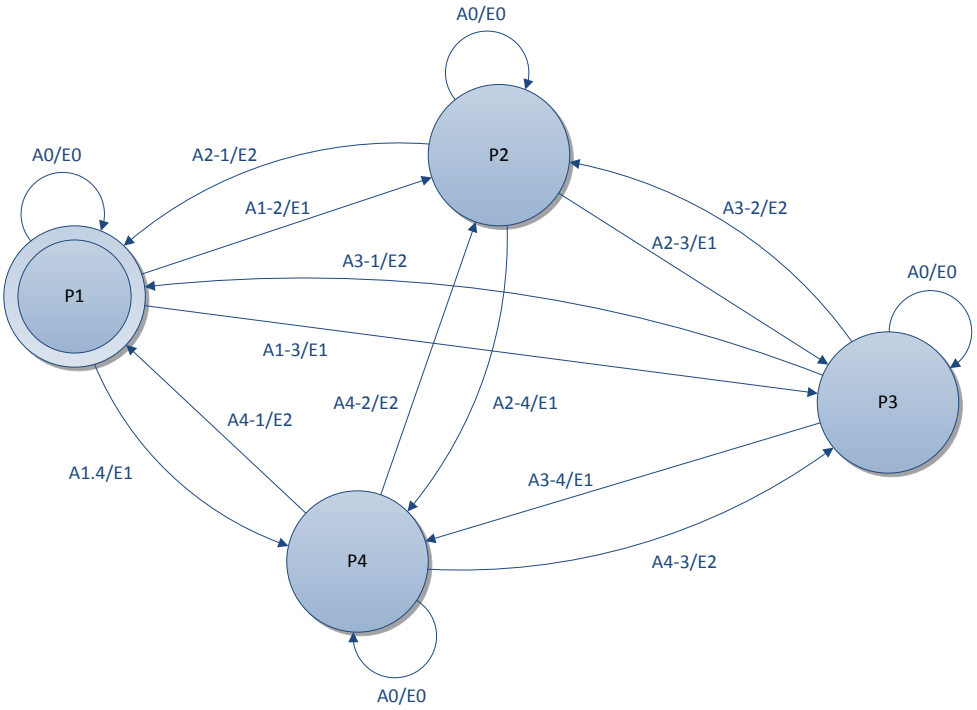


Figura 5.4: Diagrama de transiciones de estados del ascensor[1]

El funcionamiento del sistema embebido es muy sencillo. Consiste en un ascensor de cuatro pisos en el cual están implicados los siguientes estados:

- P1: Piso 1.
- P2: Piso 2.
- P3: Piso 3.
- P4: Piso 4.

Cuando el ascensor se encuentra en cualquiera de los pisos mencionados anteriormente, se encuentra en un estado de reposo esperando la siguiente acción por parte del usuario.

[1] Todos los estados son potencialmente iniciales. Al reiniciar el sistema se elige como el estado inicial P1, ya que en todo sistema secuencial debe existir un estado inicial.

Para hacer frente a estos estados, el ascensor debe responder a las siguientes acciones:

- A0: Reposo.
- A1-2: Subir al piso 2.
- A1-3: Subir al piso 3.
- A1-4: Subir al piso 4.
- A2-1: Bajar al piso 1.
- A2-3: Subir al piso 3.
- A2-4: Subir al piso 4.
- A3-1: Bajar al piso 1.
- A3-2: Bajar al piso 2.
- A3-4: Subir al piso 4.
- A4-1: Bajar al piso 1.
- A4-2: Bajar al piso 2.
- A4-3: Bajar al piso 3.

Siempre que se reproduce una acción en el ascensor, se debe atender a los siguientes eventos relacionados:

- E0: Parar el motor.
- E1: Accionar el motor para subir.
- E2: Accionar el motor para bajar.

El evento “seleccionar piso” se genera cuando se solicita que el ascensor vaya a un piso concreto, mientras que “piso encontrado” es un evento que generan los sensores que indican que se ha llegado al piso solicitado.

Para que el funcionamiento de este ascensor simplificado sea correcto se deben obedecer varias situaciones o condiciones iniciales, tales como:

- S0: Situación inicial incorrecta ya que el botón pulsado corresponde al piso actual donde se encuentra el ascensor.
- S1: Si se quiere subir, el nuevo piso debe ser mayor que el actual en caso de no encontrarse en la última planta.
- S2: Si se quiere bajar, el nuevo piso debe ser menor que el actual en caso de no encontrarse en la planta baja.

Si una combinación de un estado y un evento es legal, el estado resultante de esta combinación se incorpora en la tabla. La primera columna de la tabla contiene el estado inicial. Las otras columnas constan de los estados a los que se pueden llegar directamente desde el estado inicial (una transición desde el estado inicial). A continuación vienen los estados que están a dos de distancia del estado inicial, y así sucesivamente hasta que todos los estados están representados en la tabla. Las combinaciones de estados y eventos para una transición a sí mismo también deben ser incluidas en esta la tabla.

Para el caso práctico del ascensor, la Tabla 5.2 muestra el resultado de la tabla de estados y eventos.

Tabla 5.2: Tabla de estados y eventos del ascensor

TABLA DE ESTADOS Y EVENTOS DEL ASCENSOR				
	P1	P2	P3	P4
E1	P2, P3, P4	P3, P4	P4	x
E2	x	P1	P1, P2	P1, P2, P3
E0	P1	P2	P3	P4

Las casillas marcadas con una “x” significan que existe un camino oculto o, dicho de otro modo, que son combinaciones ilegales (casos de prueba negativos).

5.5.3.2. Árbol de transiciones

La tabla de estados y eventos se utiliza para la composición del árbol de transiciones. La Figura 5.2 se muestra un ejemplo utilizando el diagrama de estados anterior.

El estado inicial forma la raíz de este árbol. A partir de este estado inicial, se añaden todos los estados y transiciones salientes relacionadas con estos. A partir de estos estados, los siguientes niveles de estados y transiciones se colocan en el árbol de transición. Esto se repite hasta que todos los caminos alcanzan el estado final, o se alcanza de nuevo el estado inicial. Si durante la construcción del árbol se añade un estado que ya aparece en otro lugar, a continuación, el camino termina y se marca como un punto final provisional.

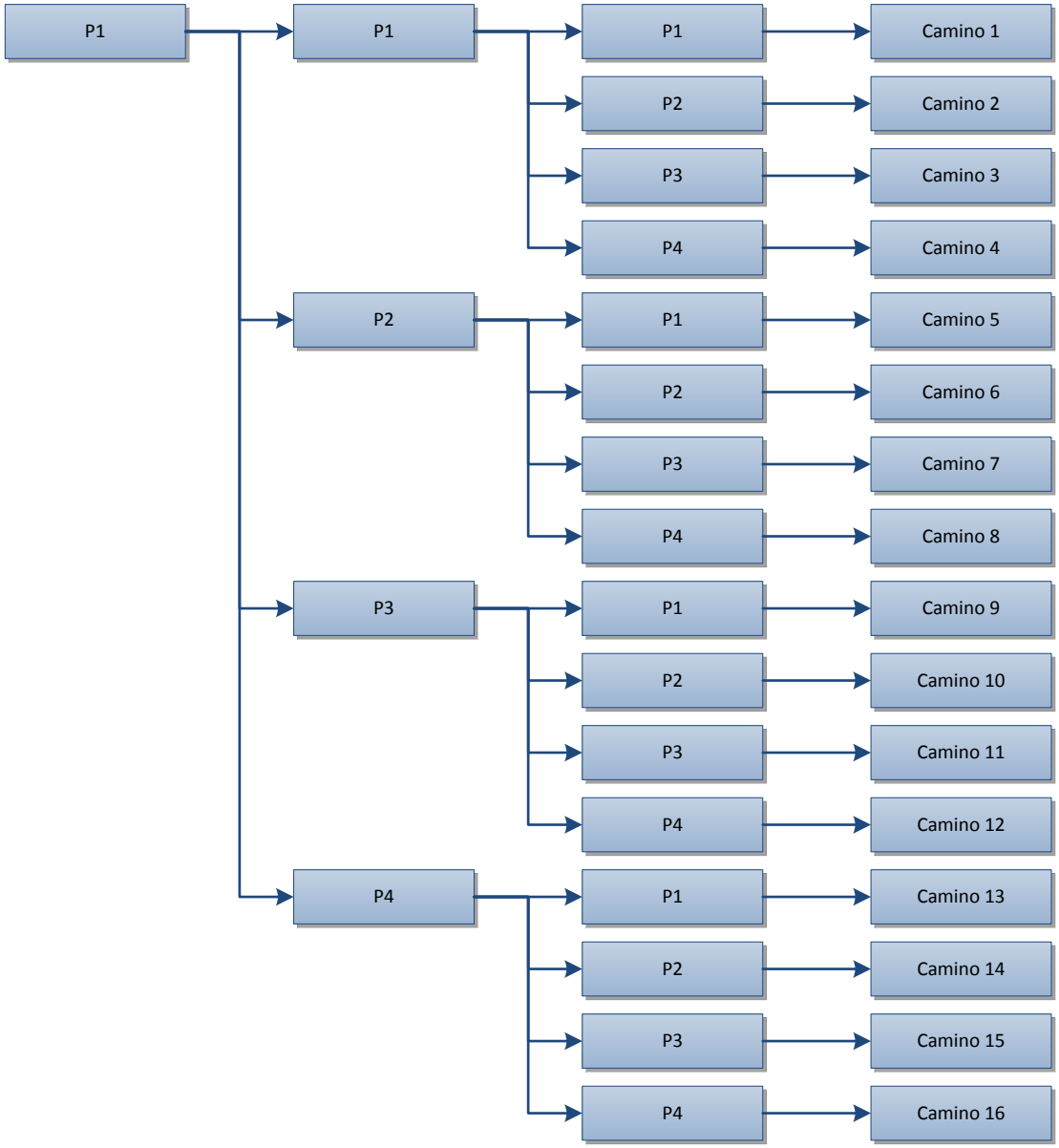


Figura 5.5: Árbol de transiciones de estados del ascensor

Si una transición a sí misma es parte del diagrama de estados, el estado resultante también se coloca en el árbol de transición. En el guion de pruebas que se obtiene del árbol de transición cada camino se completa a lo largo de la ruta más corta posible. Los caminos que no terminan en el estado inicial o final del diagrama de estados se consideran provisionales. El árbol de transiciones junto con la tabla de estados y eventos forma la descripción completa del diagrama de estados.

5.5.3.3. Guion de casos de prueba legales

Con la ayuda del árbol de transiciones y la tabla de estados y eventos, se puede crear un guion de pruebas para cubrir sólo los casos legales. La Tabla 5.3 muestra el guion de casos de prueba legales del ejemplo del ascensor.

Tabla 5.3: Guion de pruebas de los casos legales del ascensor

GUION DE PRUEBAS DE LOS CASOS LEGALES DEL ASCENSOR				
ENTRADA			RESULTADO ESPERADO	
ID	Evento	Situación inicial	Acción	Estado
CL1.1	0	0	0	1
CL1.2	1	1	1-2	2
CL1.3	1	1	1-3	3
CL1.4	1	1	1-4	4
CL2.1	2	2	2-1	1
CL2.2	0	0	0	2
CL2.3	1	1	2-3	3
CL2.4	1	1	2-4	4
CL3.1	2	2	3-1	1
CL3.2	2	2	3-2	2
CL3.3	0	0	0	3
CL3.4	1	1	3-4	4
CL4.1	2	2	4-1	1
CL4.2	2	2	4-2	2
CL4.3	2	2	4-3	3
CL4.4	0	0	0	4

Cada camino del árbol de transición es un caso de prueba. Este caso de prueba cubre todo el camino. Cada línea tiene el evento, la acción esperada y el estado resultante. Además, antes de comenzar con la realización del guion de pruebas se debe establecer la situación inicial de las pruebas para la correcta ejecución de los casos de prueba.

Para simplificar la tabla compuesta por el guion de casos legales anterior, se toma como el estado “Reposo” el valor “1”.

Para un mayor entendimiento de los casos de prueba legales se describe en la Tabla 5.4 los nemotécnicos establecidos en el enunciado del ejemplo por su equivalente descripción, es decir, una descripción detallada de los casos.

Tabla 5.4: Guion de pruebas detallado de los casos legales del ascensor

GUION DE PRUEBAS DETALLADO DE LOS CASOS LEGALES DEL ASCENSOR	
CASO LEGAL	DESCRIPCIÓN
CL1.1	Estando en el piso 1, reposar con el motor del ascensor parado debido a que se ha pulsado el botón para acceder al piso 1.
CL1.2	Estando en el piso 1, subir al piso 2 accionando el motor del ascensor en modo subida si el botón pulsado es superior al piso 2.
CL1.3	Estando en el piso 1, subir al piso 3 accionando el motor del ascensor en modo subida si el botón pulsado es superior al piso 1.
CL1.4	Estando en el piso 1, subir al piso 4 accionando el motor del ascensor en modo subida si el botón pulsado es superior al piso 1.
CL2.1	Estando en el piso 2, bajar al piso 1 accionando el motor del ascensor en modo bajada si el botón pulsado es inferior al piso 2.
CL2.2	Estando en el piso 2, reposar con el motor del ascensor parado debido a que se ha pulsado el botón para acceder al piso 2.
CL2.3	Estando en el piso 2, subir al piso 3 accionando el motor del ascensor en modo subida si el botón pulsado es superior al piso 2.
CL2.4	Estando en el piso 2, subir al piso 4 accionando el motor del ascensor en modo subida si el botón pulsado es superior al piso 2.
CL3.1	Estando en el piso 3, bajar al piso 1 accionando el motor del ascensor en modo bajada si el botón pulsado es inferior al piso 3.
CL3.2	Estando en el piso 3, bajar al piso 2 accionando el motor del ascensor en modo bajada si el botón pulsado es inferior al piso 3.
CL3.3	Estando en el piso 3, reposar con el motor del ascensor parado debido a que se ha pulsado el botón para acceder al piso 3.
CL3.4	Estando en el piso 3, subir al piso 4 accionando el motor del ascensor en modo subida si el botón pulsado es superior al piso 3.
CL4.1	Estando en el piso 4, bajar al piso 1 accionando el motor del ascensor en modo bajada si el botón pulsado es inferior al piso 4.
CL4.2	Estando en el piso 4, bajar al piso 2 accionando el motor del ascensor en modo bajada si el botón pulsado es inferior al piso 4.
CL4.3	Estando en el piso 4, bajar al piso 3 accionando el motor del ascensor en modo bajada si el botón pulsado es inferior al piso 4.
CL4.4	Estando en el piso 4, reposar con el motor del ascensor parado debido a que se ha pulsado el botón para acceder al piso 4.

5.5.3.4. Guion de casos de prueba ilegales

Las combinaciones ilegales se pueden leer a partir de la tabla de estados y eventos. Las combinaciones ilegales se refieren a aquellos casos de prueba en los que no se especifica el comportamiento del sistema para responder al evento en un estado en particular. El resultado esperado de todos los casos de prueba ilegales es que el sistema no responde.

Un caso de prueba ilegal no se debe confundir con uno en el que se especifica explícitamente que el sistema no procesa la operación y muestra un mensaje de error. El mensaje de error en este caso es, de hecho, la respuesta esperada del sistema.

La Tabla 5.5 muestra el guion de pruebas de los casos ilegales del ejemplo del ascensor.

Tabla 5.5: Guion de pruebas de los casos legales del ascensor

GUION DE PRUEBAS DE LOS CASOS ILEGALES DEL ASCENSOR				
ID	SITUACIÓN INICIAL	ESTADO	EVENTO	RESULTADO
CI1	CL1.1	1	2	Reposo
CI2	CL4.1	4	1	Reposo

Para un mayor entendimiento de los casos de prueba ilegales se describe en la Tabla 5.6 los nemotécnicos establecidos en el enunciado del ejemplo por su equivalente descripción, es decir, una descripción detallada de los casos.

Tabla 5.6: Guion de pruebas detallado de los casos ilegales del ascensor

GUION DE PRUEBAS DETALLADO DE LOS CASOS ILEGALES DEL ASCENSOR	
CASO LEGAL	DESCRIPCIÓN
CI1	Estando en el piso 1, subir al piso 2 accionando el motor del ascensor en modo bajada si el botón pulsado es superior al piso 1.
CI2	Estando en el piso 4, bajar al piso 3 accionando el motor del ascensor en modo subida si el botón pulsado es inferior al piso 4.

Se debe forzar al sistema a crear estas situaciones de prueba ilegales para comprobar que su comportamiento es el adecuado frente a situaciones extremas. Un fallo como las descritas en CI1 y CI2 puede ser catastrófico para el sistema e incluso para la

integridad física de los usuarios del sistema, en este caso la gente que utiliza el ascensor.

5.6. TECNICA DE DISEÑO: CONTROL DE FLUJO

5.6.1. Introducción

El objetivo de la prueba de control de flujo (CFT) es probar la estructura del programa. Los casos de prueba se obtienen de la estructura de un algoritmo o de un programa. Cada caso de prueba consiste en un conjunto de acciones que cubren un determinado camino por el algoritmo. La prueba de control de flujo es una técnica de diseño de pruebas que se usa principalmente en pruebas unitarias y pruebas de integración.

5.6.2. Procedimiento

La prueba de control de flujo es una técnica de diseño de pruebas que consta de los siguientes pasos:

- Construir un inventario de los puntos de decisión.
- Determinar los caminos de prueba.
- Especificar los casos de prueba.
- Establecer un conjunto de datos iniciales.
- Montar el guion de pruebas.
- Contrastar los resultados de las pruebas.

5.6.2.1. Inventario de los puntos de decisión

El diseño del programa se utiliza como la base de pruebas. Esta debe contener una descripción de la estructura del algoritmo que se quiere probar, un diagrama de flujo, una tabla de decisión, diagramas de actividad, etc. Si la base de pruebas no proporciona esta información, entonces puede ser necesario preparar la documentación de la estructura del programa sobre la base de la información disponible. Si esto no es posible, la prueba de control de flujo no se puede utilizar en este caso.

Todos los puntos de decisión en la estructura del programa deben ser identificados y etiquetados de forma exclusiva.

5.6.2.2. Caminos

La combinación de las acciones en los caminos a seguir depende del nivel de profundidad deseado. El nivel de profundidad se utiliza para decidir en qué medida se ponen a prueba las dependencias entre los puntos de decisión consecutivos. Una prueba de nivel de profundidad "*n*" significa que todas las dependencias de las acciones antes y después de un punto de decisión verifican "*n-1*" puntos de decisión. Se utilizan todas las combinaciones de acciones consecutivas "*n*".

El nivel de profundidad tiene un efecto directo en el número de casos de prueba y en el grado de cobertura de las pruebas. El efecto directo en el número de casos de prueba también significa que hay un efecto directo sobre el esfuerzo de las pruebas.

Para mostrar cómo el nivel de profundidad tiene un efecto en los casos de prueba, se da un ejemplo para una prueba con un nivel de profundidad 2. Por cada punto de decisión identificado, se determinan las combinaciones de las distintas acciones. Para un nivel de profundidad 2, se realiza un inventario de todas las combinaciones posibles de dos acciones consecutivas. En una combinación de nivel de profundidad 2 están implicadas las acciones antes y después de un punto de decisión.

Se desea realizar un cronómetro digital, que puede formar parte de un reloj o cualquier otro sistema. Su funcionamiento básico es el siguiente: al pulsar un botón empieza a contar el cronómetro y si existe una segunda pulsación la cuenta se para. Este cronómetro está limitado a mostrar una cuenta ascendente de segundos y minutos. Cuando se pulsa por segunda vez el botón, la cuenta se para y reinicia el cronómetro. El diagrama de flujo de este sistema se muestra en la Figura 5.3.

Las combinaciones de las acciones para los puntos de decisión del diagrama de flujo del cronómetro, con un nivel de profundidad 2, son:

A. (1,2); (1,4)
B. (2,3); (2,4)

Para una mayor visión de los caminos a seguir es conveniente colocar las distintas acciones en orden ascendente:

(1,2); (1,4); (2,3); (2,4)

Las combinaciones tienen que estar vinculadas para crear caminos desde el principio hasta el final del algoritmo. En este ejemplo, esto significa que cada camino tiene que empezar con la acción 1 y terminar con la acción 4.

En este diagrama de flujo es conveniente comenzar con la combinación (1,4) ya que es un camino que recorre el algoritmo de principio a fin con un nivel de profundidad 2:

(1,2); (~~1,4~~); (2,3); (2,4)

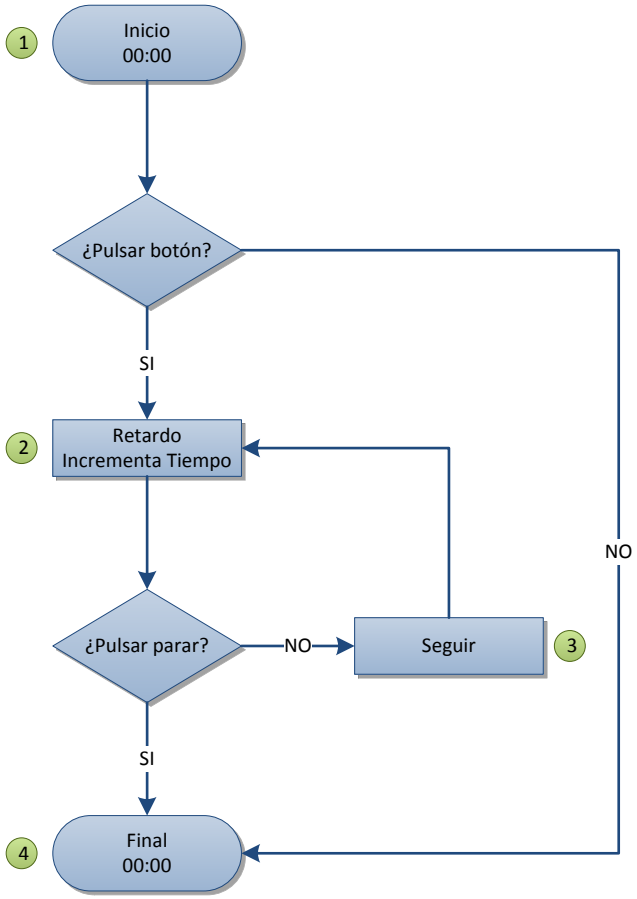


Figura 5.6: Diagrama de flujo

Se continúa con una combinación que aún no se ha incluido en un camino. En este caso, por ejemplo, se sigue por la combinación (1,2). Posteriormente, se elige una combinación que comienza por 2 y que aún no se ha incluido en el camino, por ejemplo, la combinación (2,4). Estas combinaciones se unen y se crea el camino (1,2,4). Las combinaciones restantes son:

(~~1,2~~); (~~1,4~~); (2,3); (~~2,4~~)

La última combinación se incluyen en el camino (1,2,3,2,4) y con esto, todas las combinaciones ya se han incluido en los siguientes caminos:

- Camino 1: (1,4)
- Camino 2: (1,2,4)
- Camino 3: (1,2,3,2,4)

Para una mayor comprensión de los caminos elegidos se traduce cada número por su correspondiente descripción.

Las pruebas con un nivel de profundidad 2 se basan en la idea de que la ejecución de una acción puede tener consecuencias para una acción inmediatamente después de un punto de decisión. No sólo la decisión, sino también la acción anterior son importantes. Por el contrario, en el nivel de profundidad 1 se supone que una acción sólo se ve afectada por el punto de decisión.

Un nivel de profundidad más bajo conduce a un esfuerzo más pequeño a cambio de un menor grado de cobertura.

Los niveles de profundidad más altos implican que la ejecución de una acción afecta a dos o más acciones posteriores. Los niveles altos de profundidad sólo se utilizan en partes relacionadas con la seguridad o partes muy complejas del programa.

5.6.2.3. Casos de prueba

Cada camino se convierte en un caso de prueba lógico y, a su vez, en un caso de prueba físico. Para cada camino se debe determinar una entrada de prueba. La determinación para que esta entrada sea correcta puede ser muy difícil. La entrada debe ser elegida de tal manera que en cada punto de decisión se tome el camino correcto. Si se cambia la señal de entrada durante su paso a través del algoritmo, sus valores tendrán que ser calculados de nuevo al punto de partida. A veces, el algoritmo es tan complicado que esto es prácticamente imposible, por ejemplo, cuando se utilizan algoritmos con base científica. Por cada entrada de prueba se debe especificar una predicción de la salida. Las variables y los parámetros que no tienen ningún impacto en el camino deben recibir un valor por defecto.

5.6.2.4. Conjunto de datos iniciales

A veces, la ejecución necesita un conjunto datos iniciales específicos. Esto tiene que ser descrito de una manera muy concisa. Esta descripción se añade a la descripción de los caminos y los casos de prueba.

5.6.2.5. Guion de pruebas

Todos los pasos anteriores forman la base del guion de pruebas. El guion de pruebas describe las acciones y los controles que se ejecutan en una secuencia correcta. También contiene los requisitos previos para que las pruebas sean ejecutadas con éxito. Las condiciones previas se suelen referir a la presencia de un conjunto de datos iniciales, o a que el sistema esté en un cierto estado.

Tabla 5.7: Guion de pruebas del cronómetro

GUION DE PRUEBAS DEL CRONÓMETRO	
CASO	DESCRIPCIÓN
CP1	Estando en reposo no se activa el botón de inicio
CP2	Estando en reposo se activa el botón de inicio y a continuación se para el cronómetro
CP3	Estando en reposo se activa el botón de inicio, no se para el cronómetro para seguir la cuenta y, en un momento dado, se pulsa el botón de parar

Con estos casos se cubren todas las pruebas para el sistema embebido en concreto para un nivel de profundidad 2.

5.6.2.6. Resultados de las pruebas

La ejecución de las pruebas se realiza de acuerdo a la secuencia especificada en el guion de pruebas. El resultado se compara con la predicción realizada de la salida. El siguiente paso es la detección de las posibles diferencias entre la entrada y la predicción de la salida.

5.7. TÉCNICA DE DISEÑO: COMPARACIÓN ELEMENTAL

5.7.1. Introducción

En las pruebas de comparación elemental (ECT) lo que se pone a prueba en detalle es el procesamiento del sistema. Estas pruebas verifican todos los caminos de una función o una parte del sistema. Tienen que ser identificadas todas las condiciones funcionales y deben ser traducidas a pseudocódigo. Los casos de prueba se obtienen del pseudocódigo y cubren los caminos identificados.

La ECT establece las condiciones de cobertura y garantiza un grado razonable de integridad. Debido a que la realización de esta técnica conlleva un gran trabajo, sólo se utiliza en funciones muy importantes y cálculos complejos.

5.7.2. Procedimiento

La técnica de diseño de pruebas de comparación elemental (ECT) es una técnica de diseño de pruebas que consta de los siguientes pasos:

- Analizar la descripción de la función.
- Determinar las situaciones de prueba.
- Determinar los casos de prueba lógicos.
- Determinar los casos de pruebas físicos.
- Montar el guion de pruebas.

Para mostrar estos pasos relacionados con técnica de diseño de pruebas de comparación elemental, se utiliza la siguiente descripción de una función:

```
IF SensorTemp1 ≤ 30 THEN
      Ventilador = OFF
ELSE
      IF SensorTemp2 > 30 OR SensorTemp3 ≥ 70 THEN
            Ventilador = ON
      ENDIF

      IF SensorTemp2 ≤ 30 AND SensorTemp3 < 70 THEN
            Ventilador = OFF
      ENDIF
ENDIF
```

La función consiste en la descripción del funcionamiento de tres sensores de temperatura que controlan el encendido y apagado de un ventilador. Si el primer sensor está por debajo o es igual a 30º, el ventilador se mantendrá apagado, es decir, se queda en un estado de reposo. Esto también sucede esto si el segundo sensor menor o igual a 30º y el tercer sensor es menor a 70º de temperatura. Si el segundo sensor tiene una temperatura mayor a 30º y el tercer sensor mayor o igual a 70º, el ventilador se enciende. Cuando uno de los dos sensores supera su umbral establecido es cuando se enciende el ventilador hasta volver a recuperar la temperatura establecida.

5.7.2.1. Descripción de la función

La descripción de la función debe representar los caminos de decisión y aspectos relacionados. La función puede ser descrita en pseudocódigo o por medio de otra

técnica, por ejemplo, por medio de tablas de decisión. Incluso si sólo hay una descripción, la ECT puede aplicarse si los puntos de decisión se describen de forma inequívoca. Es muy útil para traducir esta descripción funcional en pseudocódigo si no lo está ya.

El primer paso es identificar las condiciones. Estas se pueden reconocer por términos como DO, IF, REPEAT, etc. Estas están separadas una por una y se les da una identificación única (C1, C2, etc.). Se tienen en cuenta sólo las condiciones inducidas por la entrada.

```
C1:    IF SensorTemp1 ≤ 30 THEN
             Ventilador = OFF
       ELSE
C2:          IF SensorTemp2 > 30 OR SensorTemp3 ≥ 70 THEN
                   Ventilador = ON
             ENDIF

C3:          IF SensorTemp2 ≤ 30 AND SensorTemp3 < 70 THEN
                   Ventilador = OFF
             ENDIF
       ENDIF
```

5.7.2.2. Situaciones de prueba

Después de identificar las condiciones de la función que se quiere analizar, se deben determinar las situaciones que se evaluaron de cada estado. Es muy importante distinguir entre condiciones simples y condiciones complejas. Las condiciones simples constan de una sola comparación, la comparación elemental. La condición C1 es un ejemplo de una comparación elemental. Las condiciones complejas son aquellas que tienen comparaciones múltiples, conectadas por AND u OR. La condición C2 es un ejemplo de dicha condición conectada por OR y la condición C3 por AND.

El resultado de la condición simple son dos situaciones, la condición es verdadera o falsa. En el ejemplo que se está analizando, esto dará lugar a las dos situaciones.

```
C1:    IF SensorTemp1 ≤ 30 THEN
             Ventilador = OFF
       ELSE
```

El resultado de la aplicación de la ECT para la condición simple C1 sería el que se muestra en la Tabla 5.8, en la cual se comprueban las dos situaciones (se cumple o no se cumple) de la condición:

Tabla 5.8: Situaciones de prueba para la condición C1 del sensor

SITUACIONES DE PRUEBA PARA LA CONDICIÓN C1 DEL SENSOR		
SITUACIÓN DE PRUEBA	C1.1	C1.2
C1	1	0
SensorTemp1 ≤ 30	≤ 30	> 30

Una condición compleja es una combinación de condiciones simples. La condición compleja es verdadera o falsa, pero esto depende de si las condiciones simples que la forman son verdaderas o falsas. El que las condiciones estén conectadas por AND u OR es muy importante.

Tabla 5.9: Situaciones de prueba de una condición compuesta

SITUACIONES DE PRUEBA DE UNA CONDICIÓN COMPUESTA			
CONDICIÓN COMPUESTA		AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Una condición compleja con AND es cierta sólo si las dos condiciones simples son verdaderas. Mientras que una condición compleja con OR es cierta si sólo una de las dos condiciones simples es verdadera.

Las posibles situaciones para las condiciones complejas están exponencialmente relacionadas con las condiciones simples. Una condición compleja compuesta por dos condiciones simples tiene $2^2 = 4$ situaciones posibles, mientras que una condición compleja con tres condiciones simples tiene $2^3 = 8$ posibles situaciones. Para una combinación de siete condiciones simples, las posibilidades son 128 situaciones. ECT se basa en el hecho de que no es útil identificar todas estas posibilidades como situaciones de prueba. Las situaciones de prueba se seleccionan de tal manera que un cambio en el valor de cualquiera de las comparaciones elementales cambia el valor de la condición compleja. Por ejemplo, se obvian las situaciones verdadero-verdadero en el caso de una relación OR, y falso-falso en el caso de una relación AND. Las posibilidades de encontrar un defecto en las situaciones de verdadero-falso o falso-verdadero son mínimas.

En el caso de una relación AND, sólo aquellas situaciones se pondrán a prueba en la que cada comparación elemental tiene un valor verdadero de 1, y todas aquellas en las que una y sólo una comparación elemental tiene un valor verdadero de 0. Lo contrario se aplica a una relación OR, las únicas situaciones probadas son aquellas en las que cada comparación elemental tiene un valor verdadero de 0, y todos aquellos en los que una y sólo una comparación elemental tiene un valor verdadero de 1.

ECT reduce el número de situaciones de prueba al número de comparaciones elementales.

El resultado de la aplicación de la ECT para la condición compleja C2 sería el que se muestra en la Tabla 5.10:

```
C2:   IF SensorTemp2 > 30 OR SensorTemp3 ≥ 70 THEN
            Ventilador = ON
      ENDIF
```

Tabla 5.10: Situaciones de prueba para la condición C2 del sensor

SITUACIONES DE PRUEBA PARA LA CONDICIÓN C2 DEL SENSOR				
SITUACIONES	C2.1	C2.2	C2.3	C2.4
C2	0(00)	1(01)	1(10)	1(11)
SensorTemp2 > 30	≤ 30	> 30	≤ 30	> 30
SensorTemp3 ≥ 70	< 70	< 70	≥ 70	≥ 70

El resultado de la aplicación de la ECT para la condición compleja C2 sería el que se muestra en la Tabla 5.11:

```
C3:   IF SensorTemp2 ≤ 30 AND SensorTemp3 < 70 THEN
            Ventilador = OFF
      ENDIF
```

Tabla 5.11: Situaciones de prueba para la condición C3 del sensor

SITUACIONES DE PRUEBA PARA LA CONDICIÓN C3 DEL SENSOR				
SITUACIONES	C3.1	C3.2	C3.3	C3.4
C3	0(00)	0(01)	0(10)	1(11)
SensorTemp2 ≤ 30	> 30	≤ 30	> 30	≤ 30
SensorTemp3 < 70	≥ 70	≥ 70	< 70	< 70

5.7.2.3. Casos de prueba lógicos

Para determinar los casos de prueba lógicos se tienen que encontrar los caminos funcionales en los que cada situación de cada condición debe ser completada por lo menos una vez. Este enfoque puede conducir a la aparición de la misma comparación en más de una condición. Al seleccionar las combinaciones de caminos, también hay que asegurarse de que el resultado final esperado sea único, tanto como sea posible, para las combinaciones del camino elegido. Las siguientes condiciones pueden influir en los demás caminos.

Para ayudar a encontrar los casos de prueba y comprobar si se han registrado todas las situaciones de prueba se puede utilizar la Tabla 5.12. Todas las situaciones de prueba definidas aparecen en la primera columna. La segunda columna indica el valor de la condición en esta situación. Y la tercera columna contiene la siguiente condición para ser procesada. Los casos de prueba se enumeran a continuación. Si los casos de prueba se han definido correctamente, todas las situaciones de prueba se enumeraron al menos una vez.

Tabla 5.12: Casos de prueba lógicos del sensor

CASOS DE PRUEBA LÓGICOS DEL SENSOR								
SITUACIÓN	VALOR	FINAL	CL1	CL2	CL3	CL4	CL5	CONTROL
C1.1	0	Fin	X					1
C1.2	1	C2		X	X	X	X	6
C2.1	0	C3		X				1
C2.2	1	C3			X			1
C2.3	1	C3				X		1
C2.4	1	C3					X	1
C3.1	0	Fin					X	1
C3.2	0	Fin				X		1
C3.3	0	Fin			X			1
C3.4	1	Fin		X				1

5.7.2.4. Casos de prueba físicos

El paso siguiente es convertir los casos de prueba lógicos en casos de prueba físicos construidos con la ayuda de la Tabla 5.12, donde se encuentran los casos de prueba lógicos.

Estos casos de prueba físicos son muy útiles para entender y comprobar si se está realizando correctamente el análisis o si existe algún fallo en la función en sí. Se podría decir que, además de realizar los casos de prueba físicos, se está realizando una comprobación de la correcta utilización de la técnica de diseño de pruebas. Los casos de prueba físicos del sensor se muestran en la Tabla 5.13.

Tabla 5.13: Casos de prueba físicos del sensor

CASOS DE PRUEBA FÍSICOS DEL SENSOR					
CASOS DE PRUEBA	CF1	CF2	CF3	CF4	CF5
SensorTemp1	30º	31º	31º	31º	31º
SensorTemp2	X	30º	30º	31º	31º
SensorTemp3	X	69º	70º	69º	70º
Ventilador	OFF	OFF	ON	ON	ON

5.7.2.5. Guion de pruebas

El guion de pruebas deberá proporcionar toda la información necesaria para ejecutar los casos de prueba. Por lo tanto, un guion de pruebas debe contener los casos de prueba, sus acciones y controles relacionados en el orden correcto. Además, se deben enumerar todas las condiciones que se describen en la situación inicial. En la Tabla 5.14 se describe toda la información del guion de pruebas.

Tabla 5.14: Guion de pruebas del sensor

GUION DE PRUEBAS DETALLADO PARA EL SENSOR	
Caso	Descripción
CP1	Ventilador apagado con temperatura en el sensor 1 igual a 30º, independientemente del valor de los otros sensores.
CP2	Ventilador apagado con temperatura en los sensores 1, 3 y 3 igual a 31º, 30º y 69º respectivamente.

CP3	Ventilador encendido con temperatura en los sensores 1, 3 y 3 igual a 31º, 30º y 70º respectivamente.
CP4	Ventilador encendido con temperatura en los sensores 1, 3 y 3 igual a 31º, 31º y 69º respectivamente.
CP5	Ventilador encendido con temperatura en los sensores 1, 3 y 3 igual a 91º, 31º y 70º respectivamente.

5.8. TÉCNICA DE DISEÑO: ÁRBOL DE CLASIFICACIÓN

5.8.1. Introducción

Las pruebas del método del árbol de clasificación (CTM) son compatibles con el diseño sistemático de casos de prueba de caja negra. Se trata de una técnica de diseño de pruebas informal, aunque hay un procedimiento descrito para obtener los casos de prueba. CTM identifica los aspectos relevantes del sistema que se está probando. Estos aspectos que pueden influir, por ejemplo, en el comportamiento funcional o la seguridad del sistema. El dominio de entrada del objetivo de pruebas se divide en clases disjuntas utilizando particiones de equivalencia de acuerdo con los aspectos identificados. La partición del dominio de entrada se representa gráficamente en forma de árbol. Los casos de prueba están formados por una combinación de clases de diferentes aspectos. Esto se hace utilizando el árbol como la cabeza de una tabla de combinaciones en la que se marcan los casos de prueba.

Idealmente, CTM se utiliza con los requisitos funcionales como base de pruebas. Lamentablemente, estos requisitos no siempre están disponibles o completos y actualizados. Una clara ventaja de CTM es que se puede completar la información que falta. Para esto, se debe tener al menos un conocimiento básico del sistema que está probando. Si los requisitos funcionales no son los adecuados, se debe tener un conocimiento profundo sobre el comportamiento funcional deseado del sistema.

5.8.2. Procedimiento

El árbol de clasificación es una técnica de diseño de pruebas que consta de los siguientes pasos:

- Identificar los aspectos del objetivo de las pruebas.
- Partir el dominio de entrada de acuerdo con los aspectos.

- Diseñar los casos de prueba lógicos.
- Montar el guion de pruebas.

Para ilustrar estos pasos se utiliza un ejemplo que consiste en el software de control de la velocidad un motor. Los aspectos que se consideran son la velocidad y la diferencia entre la velocidad real y la velocidad deseada del motor.

5.8.2.1. Aspectos del objetivo de pruebas

El primer paso es identificar los aspectos que influyen en la transformación del sistema que se quiere probar.

Para el ejemplo, se trata de la velocidad y la diferencia de velocidad entre la velocidad actual y la velocidad deseada del motor. Estos aspectos se visualizan gráficamente en el árbol de clasificación de la Figura 5.4.

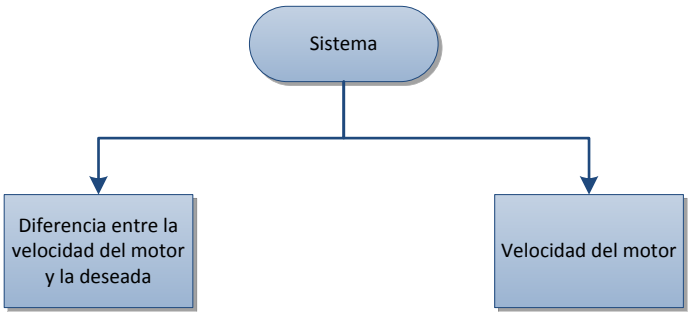


Figura 5.7: Aspectos para la velocidad de un motor

5.8.2.2. Dominio de entrada de acuerdo con los aspectos

El dominio de entrada del objetivo de las pruebas se divide en clases de acuerdo con los aspectos identificados. Por cada miembro de una clase, el sistema muestra el mismo tipo de comportamiento. Este método de partición se denomina partición de equivalencia. Las clases son disjuntas, lo que significa que cada valor de entrada no puede ser miembro de más de una clase.

La Figura 5.5 muestra el resultado del dominio de entrada, del ejemplo que se está siguiendo, de acuerdo con los aspectos indicados.

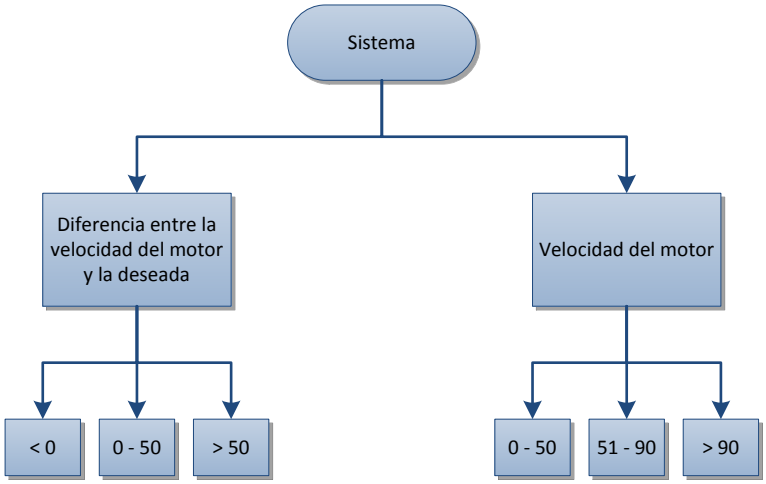


Figura 5.8: Partición del dominio de entrada del motor

Se puede dar el caso de que la creación de árbol de clasificación pueda ser una actividad recursiva. La identificación de las particiones y el dominio de entrada de acuerdo con estos aspectos se repiten hasta que se identifican todos los aspectos pertinentes y la partición del dominio de entrada esté terminada.

5.8.2.3. Casos de prueba lógicos

Una vez finalizado el árbol de clasificación ya está disponible toda la información necesaria para la preparación de los casos de prueba. Un caso de prueba está formado por la selección de una clase para cada aspecto y la combinación de estas clases.

Se alcanza el número mínimo de casos de prueba para que cada clase esté cubierta al menos por un caso de prueba. Cuando es necesario probar más a fondo, las combinaciones de clases deben ser cubiertas por los casos de prueba. La variante máxima teórica requiere que todas las combinaciones posibles de las clases de todos los aspectos deben ser cubiertos por los casos de prueba. Debido a los casos imposibles (por ejemplo, casos de prueba lógicamente inconsistentes), el número alcanzable en la práctica es, en la mayoría de los casos, más pequeño. En el ejemplo, esto significa 3 x 3 = 9 casos de prueba.

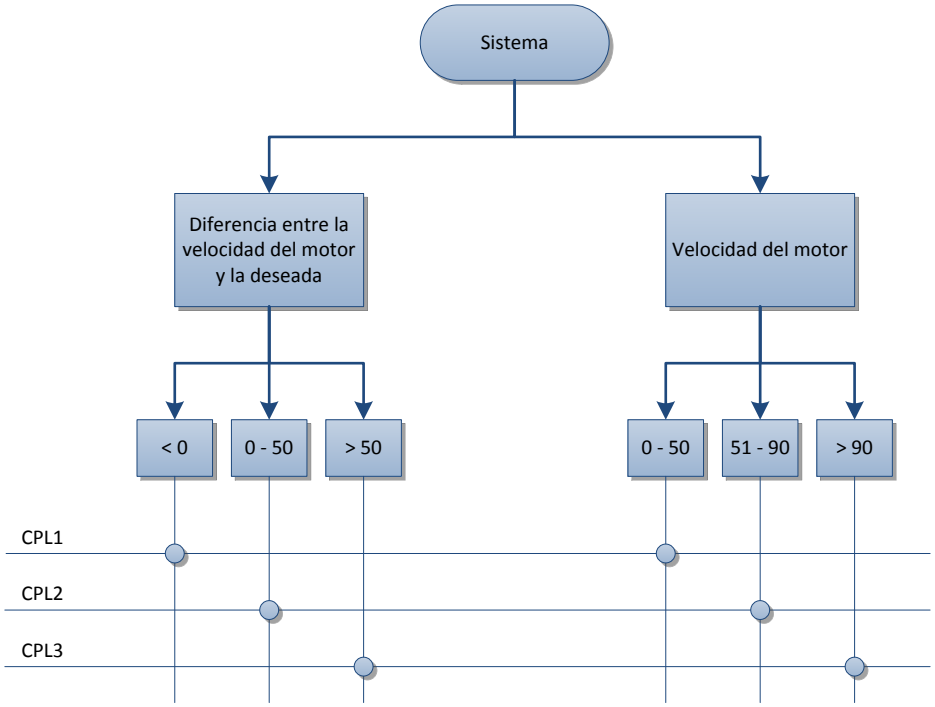


Figura 5.9: Casos de prueba lógicos para el motor

En este ejemplo, con tres casos de prueba se puede alcanzar todas las posibles combinaciones de clases de equivalencia que presenta el diagrama. Los casos de prueba obtenidos se representan a la derecha del diagrama con el número correspondiente, por ejemplo, el primer caso de prueba se representa por el número 1 y así sucesivamente.

Tabla 5.15: Casos de prueba lógicos para el motor

CASOS DE PRUEBA LÓGICOS PARA EL MOTOR		
CASO	ASPECTO	VALOR
1	Velocidad del motor Diferencia	0 - 50 < 0
2	Velocidad del motor Diferencia	51 – 90 0 - 50
3	Velocidad del motor Diferencia	> 90 > 50

Cada fila es un caso de prueba lógico, cada clase está cubierta al menos una vez con un mínimo de cuatro casos de prueba lógicos.

5.8.2.4. Guion de pruebas

La preparación de un guion de pruebas consta de las siguientes actividades: construir los casos de pruebas físicos, definir las acciones de prueba, definir los puntos de control y determinar la situación inicial.

Los casos de prueba lógicos son una combinación de clases. Con el fin de ejecutar los casos de prueba para cada clase, los valores tienen que ser elegidos para representar las clases correspondientes.

Tabla 5.16: Casos de prueba físicos para el motor

CASOS DE PRUEBA FÍSICOS PARA EL MOTOR			
CASO	ASPECTO	VALOR	COMENTARIO
1	Velocidad del motor Velocidad deseada	30 45	Diferencia entre la velocidad actual y deseada: -15
2	Velocidad del motor Velocidad deseada	65 30	Diferencia entre la velocidad actual y deseada: 35
3	Velocidad del motor Velocidad deseada	145 55	Diferencia entre la velocidad actual y deseada: 95

Una acción de prueba es una acción predefinida que puede ser exitosa o no. Para sistemas autónomos la única acción es encender el sistema. Pueden ser necesarios sistemas con una alta interacción de muchas acciones de prueba de usuario. En este último caso la descripción debe ser muy clara ya que el usuario tiene a menudo una variedad de diferentes acciones.

Un punto de control es una descripción de cuándo y cómo se comprueba el comportamiento del sistema. Se describe el comportamiento esperado para cada caso de prueba.

A veces es necesario que se cumplan algunas condiciones previas antes de que la ejecución de los casos de prueba se pueda iniciar. Estas condiciones pueden ser la disponibilidad de un determinado conjunto de datos iniciales o que el sistema tiene que ser establecido en un determinado estado. El estado y el conjunto de datos iniciales pueden variar de un caso de prueba a otro.

El guion de pruebas deberá proporcionar toda la información necesaria para ejecutar los casos de prueba. Las entregas de las actividades anteriores forman conjuntamente la información necesaria para montar el guion. Comienza con una descripción de la

situación de arranque. A continuación, las acciones de prueba y los puntos de control se describen en la secuencia correcta. Más información puede ser añadida si se necesita algún equipo de prueba adicional o se da una referencia a una lista de equipo deseado.

6. SEÑALES MIXTAS

6.1. CATEGORÍA DE LAS SEÑALES

6.1.1. Introducción

Un sistema continuo no es comparable con un sistema discreto. No hay cambios de estado y los cambios bruscos no son debidos a eventos. Todo el sistema se encuentra en un "estado" que no es exactamente definible. Para aclarar esta diferencia fundamental entre los sistemas discretos y continuos las señales se clasifican por medio de su dependencia de tiempo y valor.

En general, una señal tiene un valor que cambia con el tiempo. El cambio de tiempo y valor puede ser discreto o continuo. Como resultado de las combinaciones de tiempo y valor discreto y continuo, se pueden distinguir cuatro categorías:

- Señales analógicas.
- Señales cuantificadas en valor.
- Señales cuantificadas en tiempo.
- Señales discretas.

Las señales analógicas son continuas en el tiempo y valor. Este es el tipo físico de una señal que puede, en el dominio eléctrico, por ejemplo, ser expresada por la tensión en función del tiempo. Para la descripción de los estímulos, se puede utilizar una expresión funcional (por ejemplo, *tensión = f (t)*).

Las señales cuantificadas en valor son discretas en valor y continuas en el tiempo. Es típica de este tipo una señal analógica que pasa por un convertidor A/D para realizar cálculos posteriores en una unidad de procesamiento.

Las señales cuantificadas en tiempo son aquellas para las que el valor se conoce en sólo ciertos momentos (puntos de muestreo). Un ejemplo típico es la captura de una señal (por ejemplo, la respuesta del sistema) que está definida por los tiempos y valores pares.

Las señales digitales son aquellas cuyos valores son elementos de sólo un conjunto limitado. Si el conjunto contiene sólo dos elementos se denomina señal binaria. Debido a la forma de los datos es manejada por semiconductores, las señales binarias son de gran importancia para el procesamiento de señales y control interno en sistemas embebidos.

6.1.2. Señales analógicas puras

Existe una diferencia fundamental entre la definición de las señales de entrada (estímulos) y respuestas esperadas, por una parte, y los resultados capturados por la otra. Las señales pueden ser especificadas por ecuaciones algebraicas (por ejemplo, *f(x) = sin(x)*), que representa el comportamiento continuo de una señal sinusoidal. Sin embargo, una captura de una señal siempre se realiza por muestras y es, por lo tanto, discreta. Por otra parte, la separación del ruido de una señal capturada, que también es continua, es mucho más compleja en comparación a las señales digitales. En consecuencia, es imposible capturar una señal analógica con exactitud. Siempre hay una incertidumbre entre los puntos de muestreo. Debido al valor de la continuidad (no sólo dos, sino un número infinito de elementos pertenecen al rango), el valor exacto, esperado, o definido rara vez se puede realizar en un entorno físico. Incluso con un simulador que utiliza un muestreo variable para el cálculo de algoritmos complejos, por ejemplo, ecuaciones diferenciales algebraicas (DAE), el resultado exacto no es predecible. Por lo tanto, las tolerancias y la definición del rango de una señal tienen que utilizarse al decidir si dos señales pueden ser consideradas “iguales”. Esto es importante, por ejemplo, en el establecimiento de los criterios de los resultados de las pruebas, es decir, si se pueden dar por válidos o no.

La principal dificultad que surge en las pruebas de componentes analógicos es que los fallos analógicos no causan un cambio de estado simple en el valor lógico. Por lo tanto, la decisión sobre si se ha producido un fallo o no, no es simplemente una cuestión de "HI" y "LO". Son necesarios límites (tolerancias en el tiempo) y bandas (tolerancias en el valor).

6.1.3. Señales mixtas

El manejo de señales digitales es claro, una señal es o HI o LO, en consecuencia, es claramente definible y cuantificable. Los fallos pueden ser detectados fácilmente y los cálculos ejecutados sin ambigüedad. Tratar con señales analógicas es diferente. Pero la característica definitiva de los sistemas con señales mixtas es la combinación de controladores digitales, ordenadores y subsistemas modelados como autómatas finitos junto con controladores y plantas modeladas por ecuaciones diferenciales parciales u ordinarias o ecuaciones en diferencias. Por lo tanto, el manejo de señales mixtas, y en especial la sincronización de los dos dominios (digital y analógico), es un tema muy importante.

La combinación de señales analógicas y digitales es extremadamente difícil para el procesamiento de una señal interna. Por lo tanto, en el mundo del diseño, el tiempo se controla de forma diferente desde el mundo analógico y digital. Por ejemplo, se utilizan dos algoritmos de paso temporal diferentes para la resolución de ecuaciones durante la simulación. La parte digital, basada en una señal de reloj, se ejecuta con un tamaño del muestreo que define los puntos exactos del tiempo en los cambios de valores y estados, mientras que para la solución DAE que describen la parte analógica es necesaria una base de tiempo continuo. Ahora bien, si se produce un evento en la parte digital que influye en la parte analógica, debe llevarse a cabo una armonización, es decir, los dos instantes diferentes tienen que sincronizarse. Se puede lograr esto retrocediendo el plazo de ejecución o esperando un tiempo posterior. Además, también tienen que estar sincronizadas cuando se producen umbrales pasajeros o impulsos que inician influencias del mundo analógico al digital.

Todos estos aspectos específicos del dominio de la señal mixta necesitan medidas de prueba ajustadas y especializadas.

6.2. TÉCNICAS DE DESCRIPCIÓN DE ESTÍMULOS

6.3. Introducción

Para un sistema embebido, las descripciones de los estímulos son un conjunto de señales analógicas o digitales que se componen de varios rangos. Una señal digital está basada en una señal de reloj. En relación con el reloj, pueden ser definidos los retardos de tiempo explícitos para flancos de subida y bajada. Por otra parte, puede ser fijado un formato para un cambio concreto en el valor (por ejemplo, de acuerdo con la señal de reloj anterior). Para una señal analógica pura, la descripción se puede basar en una expresión simbólica, offset, desplazamiento de fase y la amplitud, además de la expresión de la función. Por último, los estímulos realistas para el complejo sistema que se quiere probar requieren la descripción secuencial de varios rangos de señales definidas de forma independiente.

Para todos estos niveles de prueba, se utilizan señales de prueba (a menudo llamadas patrones). La descripción pura de la señal (por ejemplo, cambios en los valores lógicos o descripciones algebraicas de señales continuas) es la misma para todos los niveles de prueba. Como consecuencia de ello, los patrones pueden ser reutilizados durante las etapas de prueba, por ejemplo, SIL (Software In Loop) HIL (Hardware In Loop), y en un desarrollo posterior. También, los resultados de las pruebas de los diferentes

niveles pueden ser comparados y reutilizados como “resultados esperados” para la etapa posterior.

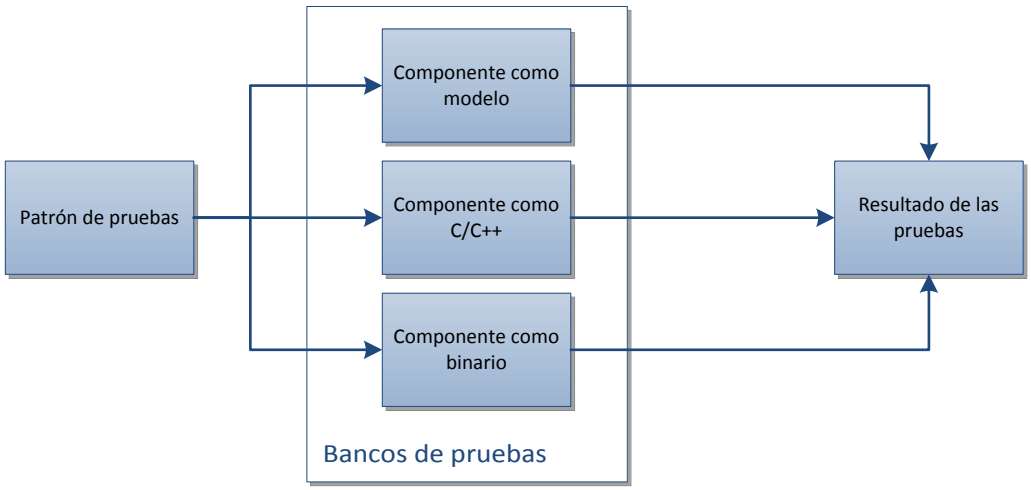


Figura 6.1: Reutilización de la entrada y la salida en los diferentes niveles

Así que la reutilización de las descripciones lógicas de las entradas y las salidas es posible, pero la representación física de las señales es diferente para los distintos niveles de prueba. El flujo de datos puro es suficiente cuando se prueban los modelos con SIL. Pero para HIL, la tensión y la corriente se deben especificar en términos concretos.

6.3.1. Criterios de evaluación y selección

Existen métodos y herramientas para probar estos sistemas que utilizan técnicas diferentes de descripción de estímulos. A pesar de que las pruebas de funcionamiento son de gran importancia para la verificación de los sistemas y de que esté generalizada en la industria, hay muy pocas técnicas y herramientas para la generación sistemática y metodológica basadas en las descripciones de estímulos correspondientes. En muchos casos, por lo tanto, se utilizan las herramientas y notaciones de una aplicación limitada.

Con el fin de apoyar la elección de un problema específico de la técnica de descripción de estímulos o una herramienta adecuada para un proyecto concreto, se necesitan criterios adecuados. Algunos ejemplos de los criterios en la descripción de las herramientas y las técnicas son los siguientes:

- Categorías de señales compatibles.
- Tipo de notación.

- Grado de abstracción.
- Complejidad.
- Consideración de los resultados.
- Criterios de cobertura.
- Soporte metodológico.
- Herramientas de apoyo.
- Formato de almacenamiento de datos.

Con el fin de mostrar la variedad de las técnicas de descripción de estímulos y herramientas existentes, y para demostrar la aplicación de la mencionada serie de criterios, se describe una selección de técnicas.

6.3.2. Cronogramas

Los cronogramas permiten la representación gráfica de varias señales discretas por medio de formas de onda simbólicas sobre un eje de tiempo global, así como las relaciones entre ellos (por ejemplo, retardos). Los estímulos, así como los resultados esperados, se pueden especificar y relacionar entre sí. Existen diferentes tipos de cronogramas. Los cronogramas se aplican, por ejemplo, a la descripción gráfica de los bancos de pruebas para los modelos de circuitos VHDL o para la descripción de las propiedades del modelo durante la comprobación del mismo.

El lenguaje del cronograma (TDML) proporciona un formato de intercambio para el conjunto de la información desde el cronograma y diferentes formatos de forma de onda. El TDML basado en XML permite el almacenamiento de esta información de una forma estructurada para que pueda ser fácilmente procesado por diferentes herramientas.

En la Tabla 6.1 se muestra una evaluación de la técnica de descripción de estímulos TDML:

Tabla 6.1: Evaluación de la técnica de descripción de estímulos TDML

EVALUACIÓN DE LA TÉCNICA TDML		
ESTÍMULOS	EVALUACIÓN	DESCRIPCIÓN
Categorías de las señales compatibles	-	Foco principal de las señales digitales
Tipo de notación	+	Textual y gráficamente (TDML)

Grado de abstracción	+	Abstracto
Complejidad	0	Secuenciación, sin necesidad de programación, se pueden describir las dependencias entre las señales
Consideración de los resultados	+	
Criterios de cobertura	-	
Soporte metodológico	-	
Herramientas de apoyo	+	Disponibles comercialmente
Formato de almacenamiento de datos	0	TDML: publicado, no ha sido estandarizado

6.3.3. Árbol de clasificación

El árbol de clasificación es un poderoso método para las pruebas de caja negra. Una notación del método para sistemas embebidos (CTM/ES) permite la descripción de estímulos para los sistemas con señales mixtas.

La descripción de estímulos tiene lugar en dos niveles diferentes de abstracción: en el primer nivel se realiza una descripción abstracta de los estímulos que, sin embargo, deja abierto ciertos grados de libertad. En esta etapa, la descripción de estímulos todavía no se limita a una sola señal. En su lugar, se especifica un conjunto de posibles señales. Los grados de libertad se eliminan en el segundo nivel, en otras palabras, se selecciona una única señal de estímulo a partir del número de posibles secuencias de entrada.

Para la descripción lógica abstracta de los estímulos, se clasifica y se visualiza el dominio de entrada del sistema por medio de una representación en forma de árbol, llamado árbol de clasificación. Este es obtenida de la interfaz del sistema definido por el “repertorio lingüístico” para la descripción de los escenarios de pruebas en la matriz subyacente, el denominado cuadro de combinaciones. Aquí todos los cambios temporales de las señales de entrada se describen de una manera abstracta.

Los estímulos están representados en el árbol por nodos rectangulares llamados clasificaciones. Para cada señal se determinan los valores admisibles y se dividen en clases de equivalencia (intervalos o valores individuales).

Los bloques de construcción de las descripciones de estímulos abstractos son pasos de prueba. Estos comprenden, de acuerdo con su orden temporal, las filas de la tabla de combinaciones. Tal secuencia de pasos de prueba se llama secuencia de pruebas. Cada paso de prueba define las entradas del sistema durante un cierto tiempo. Los intervalos de tiempo se enumeran en una columna separada en el lado derecho de la tabla de combinaciones. Los puntos de inicio y fin de estos intervalos de tiempo se denominan puntos de sincronización, ya que se sincronizan las señales de estímulos al principio y al final de cada paso de prueba.

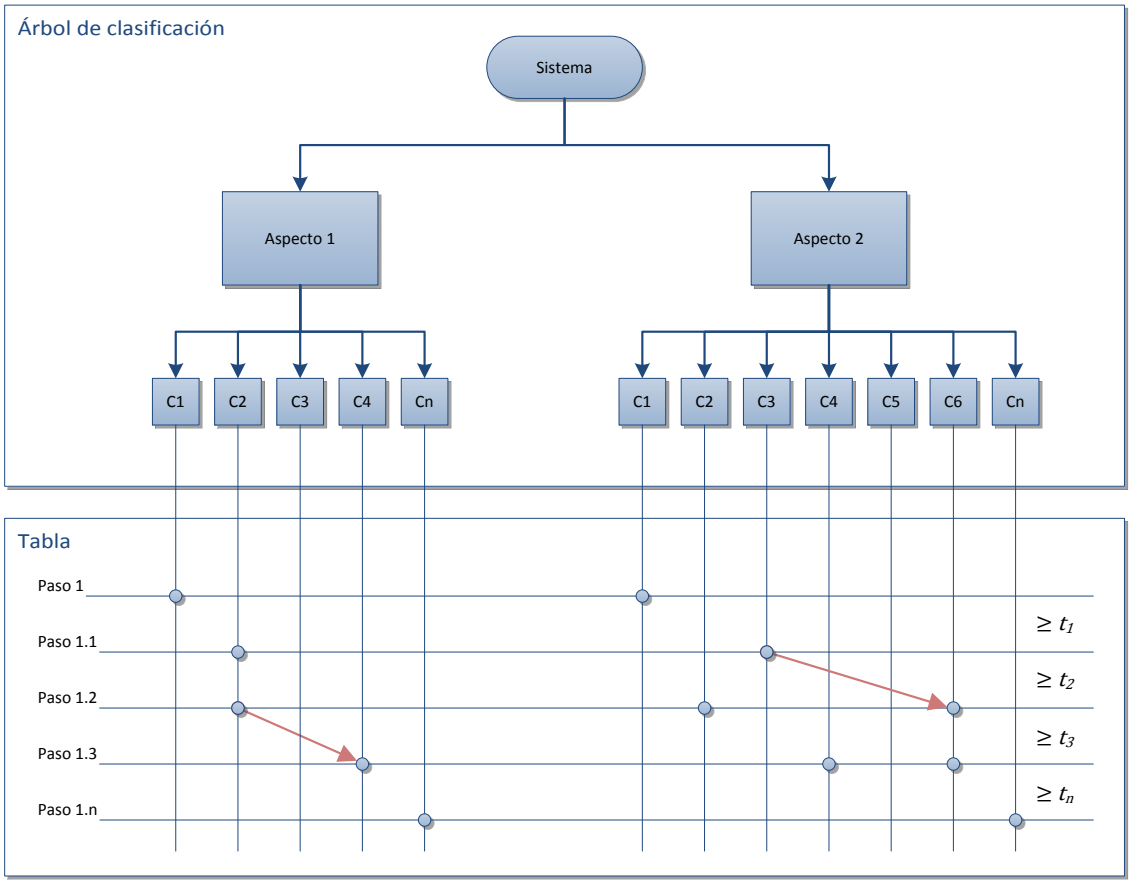


Figura 6.2: Descripción abstracta de los escenarios de prueba con CTM / ES

La descripción de los valores de las señales de estímulos individuales para cada paso de prueba se realiza mediante el marcado en el árbol de la clase definida para esta señal. Esto se indica en la parte media de la tabla de combinaciones. La señal de estímulo, en el paso de la prueba correspondiente, se reduce a la parte del intervalo o del valor individual de la clase marcada. La combinación de las clases de entrada

marcadas de un paso de prueba determina la entrada del sistema en el respectivo punto de apoyo.

Los valores de las señales de estímulos individuales entre los puntos de sincronización se describen por las formas básicas de la señal. Diferentes formas de señales (por ejemplo, rampa, función escalón, sinusoidal) están representadas por diferentes tipos de flechas que conectan dos marcas sucesivas en el cuadro de combinaciones. De este modo, las señales de estímulos analógicas y de valor cuantificado se pueden describir de forma abstracta, con la ayuda de las funciones definidas por etapas, con parámetros.

Los grados de libertad restantes son eliminados por la selección de los datos de prueba concretos. La discretización de los valores de la señal sigue para que el tiempo cuantificado o las señales digitales estén disponibles.

La especificación abstracta de las señales de entrada en el árbol de clasificación también se puede entender como tubos de señal o corredores de señal en el que deben ajustarse los cursos reales de las señales. Dado que la estimulación de los objetivos de las pruebas se llevan a cabo discretamente en el tiempo, debe ser especificada la frecuencia de muestreo con la que se discretizan los cursos de las señales. El resultado de la discretización es una descripción concreta de estímulos por pares de tiempo-valor.

6.4. TÉCNICAS DE MEDICIÓN Y ANÁLISIS

6.4.1. Introducción

La comparación de las señales capturadas en la salida del sistema (denominado como *f '(t)*) con los resultados esperados (*f(t)*) es una de las técnicas más utilizadas. Una alternativa es calcular los valores característicos. Estos métodos de análisis son adecuados para la detección de determinadas categorías de fallos, éstos se pueden clasificar de la siguiente manera:

- Fallos funcionales.
- Fallos estadísticos.
- Fallos sistemáticos.

Es posible combinar varias clases de fallos. Además, todos los fallos pueden aparecer todo el tiempo (a nivel global), o sólo en ciertos intervalos de tiempo (a nivel local).

6.4.2. Comparación de las señales esperadas y capturadas (tolerancias)

Para validar una señal capturada se pueden ser utilizadas varias técnicas con la ayuda de una de referencia.

En general, cuando se compara una señal capturada con una esperada se calcula algún tipo de diferencia definida. Debido al hecho de que ningún método puede garantizar la captura de una reproducción exacta de una respuesta del sistema, se necesitan definiciones de tolerancia. Al hacer esto, se realiza una distinción entre el error absoluto y relativo de acuerdo a la tolerancia.

$$|f(x) - f'(x)| \leq tol_y \quad (error\ absoluto)$$

$$|f(x) - f'(x)| \leq \Delta_y \quad (error\ relativo)$$

Esta definición simple de tolerancia, en relación sólo con el valor, implica algunos problemas graves. En el caso de cambios dramáticos en los valores de la señal en un pequeño periodo de tiempo, un tubo de tolerancia definido solamente por el valor en un cierto punto en el tiempo dará lugar a muchos errores observados si la tolerancia no se considera así. En otras palabras, cuando se producen pequeños cambios en el tiempo, los niveles de tolerancia se violan fácilmente si el gradiente de la señal es alto.

Por esta razón, la comparación de la señal tiene que considerar las tolerancias en el tiempo "*y*" y "*t*", así como en valor.

6.4.3. Análisis de los valores característicos

En lugar de comparar todos los valores muestreados de las dos señales, la comparación se puede reducir a unos pocos puntos importantes, esto reduce el esfuerzo de las pruebas de manera significativa. El paso de un umbral, como uno de tales puntos significativos, puede ser el indicador para una comparación de la señal en un solo punto.

- Mínimos y máximos globales y locales.
- Valores de amplitud media (50%).
- Puntos de inicio y final (10% / 90%) de aumento y caída de la señal.

6.4.4. Correlación

La correlación puede ayudar a comparar la señal capturada con la señal esperada al detectar similitudes y cambios entre estas señales. En particular, la correlación cruzada es el cálculo que proporciona los resultados más significativos:

$$z(k) = \sum_{i=0}^{N-1} x(i)h(k+1)$$

La señal de referencia es *x(i)*, la capturada *h(i)* y *z(k)* es el resultado, *k* es el cambio entre estas dos señales. El problema con este cálculo es que se lleva a cabo para toda la señal. Esto oculta los errores locales y proporciona la validación sólo de la señal como un todo.

Por lo tanto, es útil separar las señales en fases de evaluación en diferentes áreas. Estas áreas deben cubrir un período tan corto como sea necesario para obtener resultados significativos.

Alternativamente, el resumen final se queda fuera. Además, no hay multiplicación de pares de muestras pero en su lugar se realiza una resta. El resultado es entonces una matriz de extenderse por *k* e *i*. En esta matriz, la tercera dimensión está dada por el valor de la diferencia. Una indicación digital (-1, 0, 1) de esta diferencia se obtiene de la siguiente fórmula:

$$temp = abs[refsignal(i) - simsignal(i+\tau]$$

$$z(\tau, i) = \begin{cases} -1 & si\ la\ muestra\ es\ irrelevante \\ 0 & temp > tolerancia \\ 1 & temp \leq tolerancia \end{cases}$$

6.4.5. Relación señal-ruido

La relación señal-ruido de una señal viene dada por la expresión que se muestra a continuación:

$$S = 20dB \cdot log \frac{P_{signal}}{P_{Noise}}$$

Esta fórmula se utiliza cuando la señal de ruido puede ser capturada y la señal deseada es también conocida. Uno de los dominios de aplicación es en la conversión A/D. La relación señal-ruido no es un indicador de la pérdida de información por parte de la cuantificación.

6.4.6. Distorsión armónica

La distorsión armónica total, *k*, es la relación de la superficie eficaz para el valor eficaz total, incluyendo *t*,

$$k = 100\% \sqrt{\frac{U_2^2 + U_3^2 + \cdots}{U_1^2 + U_2^2 + U_3^2 + \cdots}}$$

THD (Total Harmonic Distortion) es un número característico para las distorsiones no lineales de una señal. Para capturar este tipo de errores, una señal sinusoidal definida estimula al sistema y se evalúa el resultado. El número de ondas armónicas adicionales es un indicador de la distorsión. THD se utiliza principalmente en alta frecuencia y aplicaciones de audio.

7. SEGURIDAD

7.1. IMPORTANCIA DE LA SEGURIDAD

7.1.1. Introducción

La seguridad es la expectativa de que un sistema, en condiciones conocidas, conduzca a un estado en el que se pone en peligro la vida humana.

Muchos sistemas embebidos se utilizan en situaciones críticas para la seguridad. El mal funcionamiento puede tener consecuencias graves, como lesiones o daños ambientales, o incluso la muerte. El proceso de diseño y desarrollo de estos sistemas deben tener algunas medidas integradas que impidan que sea una amenaza para la seguridad.

Para un mayor control de que se cumplen los requisitos de seguridad del sistema, se debe iniciar la supervisión en durante el diseño. Las dos técnicas de análisis de seguridad más usadas son el análisis del modo de fallos y efectos (FMEA) y el análisis del árbol de defectos (FTA).

En el análisis de la seguridad se utilizan las categorías de severidad. Estas categorías facilitan la clasificación de los riesgos identificados. Esta clasificación puede utilizarse para formular una estrategia en la que se toman diferentes medidas para diferentes categorías de riesgo.

7.1.2. Relación causa y efecto

Para analizar la seguridad se debe estudiar la relación que existe entre la causa y el efecto. El efecto siempre está relacionado con la puesta en peligro de la vida.

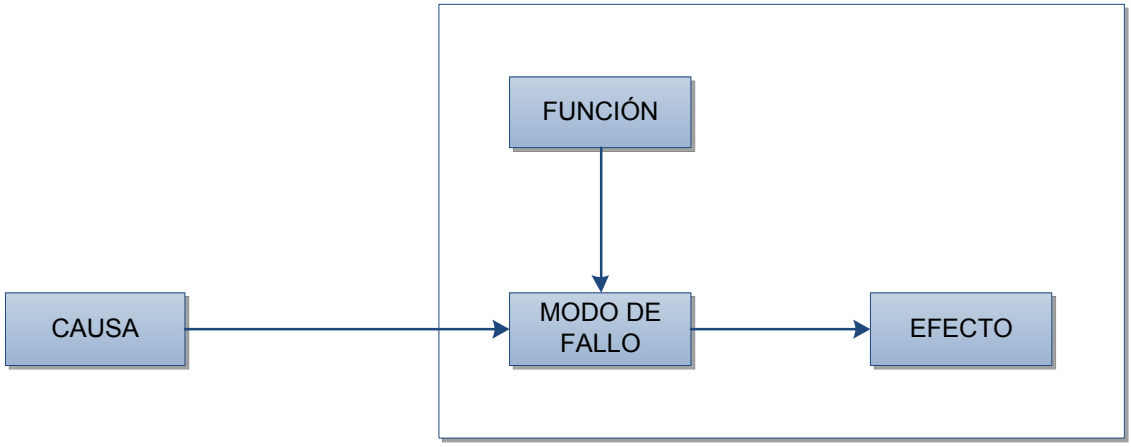


Figura 7.1: Relación entre causa, función, modo de fallo y efecto

Las causas de un fallo son muy variadas. A continuación se describen algunas causas típicas de fallos en los sistemas embebidos relacionados con la seguridad:

- Desgaste.
- Condiciones ambientales, como la interferencia electromagnética, mecánica, química y perturbación.

El modo de fallo describe la forma en que un producto o proceso puede fallar para realizar su función deseada descrita por las necesidades, deseos y expectativas de los clientes internos y externos. Un fallo es la incapacidad de un sistema o componente para cumplir sus necesidades operativas. Los fallos pueden ser sistemáticos o debidos a un cambio físico. Un efecto es una consecuencia adversa causada por un modo de fallo.

7.2. TÉCNICAS DE ANÁLISIS

7.2.1. Introducción

En primer lugar, se debe definir el alcance del análisis. Para permitir un análisis más concreto acerca de lo seguro que algo debe ser, se construye una tabla de clasificación de riesgo. En segundo lugar, se pueden aplicar las técnicas de análisis FMEA y FTA.

7.2.2. Análisis del modo de fallo y los efectos

FMEA es un método de análisis ascendente que determina el efecto de un modo de fallo (datos corruptos o comportamiento inesperado) en el sistema. La técnica se utiliza al principio del proceso de diseño, cuando es más fácil tomar las medidas necesarias para superar los problemas identificados, lo que mejora la seguridad a través del diseño. FMEA comprende tres etapas: identificar los modos de fallo potenciales, determinar los efectos de estos modos de fallo potenciales en el funcionamiento del sistema y formular acciones para disminuir los efectos.

El uso adecuado de FMEA puede tener algunos beneficios, como por ejemplo, la mejora de la seguridad del sistema, el seguimiento de los riesgos durante el ciclo de vida del desarrollo, la identificación de los riesgos potenciales de seguridad, etc.

Un típico FMEA se lleva a cabo como sigue:

- Se describe el sistema y sus funciones.
- Se identifica el modo de fallo potencial.
- Se describen todas las funciones con respecto a los efectos.
- Se identifican las causas de los efectos y los riesgos.
- Se realizan pruebas sobre los riesgos identificados.

7.2.3. Análisis del árbol de defectos

FTA se usa para identificar las causas de los fallos. Es una técnica para analizar el diseño en cuanto a seguridad y fiabilidad. El fallo del sistema se representa en la parte superior del árbol de fallos y el siguiente paso es considerar qué comportamiento no deseado, de las partes del sistema, es responsable del mal funcionamiento de este. En este contexto, un fallo del sistema es un evento incorrecto, datos incorrectos, datos inesperados o un mal comportamiento.

Como se dijo anteriormente, el fallo del sistema se coloca en la parte superior del árbol de fallos. El siguiente paso es determinar qué causó ese fallo. Cada etapa siguiente, depende de la causa del fallo del paso anterior. Este análisis lleva a la causa del fallo del sistema. Los requisitos de seguridad son la base para decidir lo que es inesperado o no deseado en el comportamiento del sistema. FTA es un trabajo para un equipo de expertos.

7.3. CICLO DE VIDA DE LA SEGURIDAD

7.3.1. Introducción

La construcción de un sistema crítico para la seguridad consiste en seguir lo que dicta la ley y las autoridades de certificación. Algunas de las normas relacionadas con estos sistemas son muy estrictas. A fin de cumplir estas normas tiene que haber un proceso estructurado con resultados claros. Existe un estándar para la gestión de la seguridad llamado “MOD-00-56” (MOD, 1996a, 1996b). Parte de este estándar es la descripción de los procesos a desarrollar e implementar un sistema crítico para la seguridad.

El ciclo de vida es una descripción estructurada de un diagrama de flujo. El objetivo de este proceso consiste en pasar de unos requisitos globales a un sistema certificado en seguridad. Todos los diseños, decisiones y resultados del análisis se almacenan en un registro de posibles riesgos. Esto forma la base para el estudio de la seguridad.

- Requisitos de seguridad.

- Programa de seguridad del proyecto.
- Identificación de los riesgos.
- Estimación de los riesgos.
- Evaluación del cumplimiento de la seguridad.
- Comprobación de la seguridad.

Una descripción detallada de este proceso y todas las actividades se pueden encontrar en el estándar “MOD 00-56” (MOD, 1996a, 1996b).

7.3.2. Base de pruebas

El diseño final es parte de la base de pruebas y también de las pruebas de seguridad. El diseño final se realiza durante un proceso iterativo. El proceso se inicia con un diseño global. Sobre esta base se ejecuta la identificación y la estimación del riesgo. El resultado de la estimación de los riesgos puede ser que se corrijan algunas acciones, de lo contrario, el diseño global se elabora con más detalle y, a continuación, se vuelve a ejecutar el proceso. Este continúa hasta que no haya más acciones que corregir y el diseño está listo.

El diseño final junto con los requisitos (funcionales, rendimiento, facilidad de uso, etc.) es la base de pruebas.

7.3.3. Actividades

En la ejecución de las pruebas, en el proceso de pruebas y en el proyecto de seguridad se producen defectos. Estos deben ser analizados y corregidos. La corrección de un defecto de seguridad puede influir en el funcionamiento del sistema, y viceversa, una corrección de un defecto funcional puede afectar a la seguridad. Con el fin de superar este problema se deben centralizar el análisis de impacto y las acciones correctivas deben ser centralizadas.

8. INFRAESTRUCTURA

8.1. ENTORNO DE PRUEBAS

8.1.1. Introducción

El desarrollo de un sistema embebido implica muchas actividades de pruebas en las diferentes etapas del proceso de desarrollo, cada uno requiere instalaciones específicas para cada prueba.

Para el desarrollo y las pruebas de un sistema embebido existen varias etapas relacionadas con el modelo en V:

- Simulación.
- Prototipo.
- Pre-producción.
- Post-desarrollo.

Los primeros modelos desarrollados para simular el sistema se prueban con “modelos de prueba” (MT) y "modelos en bucle" (MIL), esto pertenece a la etapa de simulación. En “prototipo rápido” (RP), un modelo experimental con capacidad de alto rendimiento y un montón de recursos (por ejemplo, procesamiento de punto flotante de 32 bits) se prueba en un entorno de la vida real para comprobar si el sistema puede cumplir su objetivo, esto es también parte de la etapa de simulación. En las pruebas de “software en bucle” (SIL), el software real, incluyendo todas las restricciones de recursos (por ejemplo, el procesamiento de número entero de 16 bits o 8 bits), se prueba en un entorno simulado o con el hardware experimental. En “hardware en bucle” (HIL), el hardware real es utilizado y probado en un entorno simulado. Tanto "SIL" y "HIL" forman parte de la fase del prototipo. En las “prueba del sistema” (ST) del sistema real se prueba en su entorno real, lo que corresponde a la etapa de pre-producción.

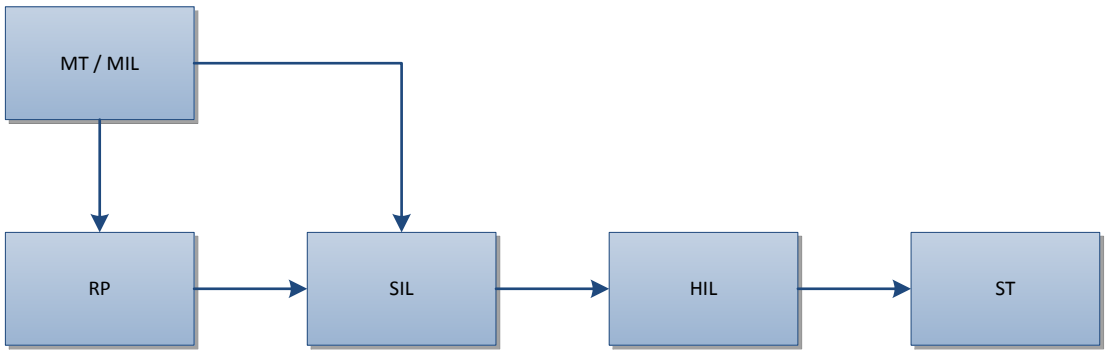


Figura 8.1: Proceso de pruebas desde la simulación a la situación real

8.1.2. Primera etapa: simulación

En el modelo de desarrollo basado en la comprobación de los requisitos y el diseño conceptual, se construye un modelo de simulación. El desarrollador de un sistema embebido utiliza los modelos de simulación ejecutables para apoyar al diseño inicial, para probar el concepto y para desarrollar y verificar los requisitos detallados para los próximos pasos del desarrollo. Esta etapa también se describe como "modelo de prueba" y "modelo en bucle". Los objetivos de las pruebas en esta etapa son probar los requisitos y optimizar el diseño.

La ejecución de un modelo de simulación y el control de su comportamiento requiere un entorno de pruebas específico. Se necesitan herramientas software dedicadas a generar y ejecutar el modelo de simulación, para generar señales y analizar sus respuestas.

Las pruebas que se realizan en esta primera etapa de simulación consisten, en general, de los siguientes pasos:

- Simulación individual.
- Simulación realimentada.
- Prototipo rápido.

El modelo del sistema embebido se prueba de forma aislada. Se introduce de una en una las entradas en el sistema simulado y se analiza la salida resultante. La interacción dinámica con el entorno se tiene en cuenta.

Se pone a prueba la interacción entre el sistema embebido simulado y la planta. Otro término para esto (que se utiliza en los sistemas de control de procesos) es que son comprobadas las “leyes de control”. El modelo de la planta genera una entrada para el sistema embebido. La salida resultante se alimenta de nuevo a la planta, lo que resulta una nueva entrada para el sistema embebido, y así sucesivamente.

El sistema embebido simulado se prueba mientras está conectado al entorno real. Esta es la mejor manera de evaluar la validez del modelo de simulación.

8.1.3. Segunda etapa: prototipos

El desarrollo de un sistema embebido sin el uso de un modelo comienza en este punto. Al igual que en el caso del desarrollo basado en el modelo como se describe

anteriormente, la etapa de creación de prototipos se inicia después de que se han alcanzado los objetivos de la etapa de simulación.

Los objetivos de las pruebas en esta etapa son demostrar la validez del modelo de simulación, comprobar que el sistema cumple con los requisitos y liberar la unidad de pre-producción.

En esta etapa, el hardware y las versiones experimentales y reales del software reemplazan gradualmente a los componentes simulados. Se presenta la necesidad de interfaces entre el modelo de simulación y el hardware. Se pueden utilizar emuladores. Las señales que estaban disponibles en el modelo de simulación pueden no ser lo más accesible con el hardware real. Por lo tanto, pueden tener que ser instalados transductores de señal especiales, así como la grabación de señales y equipos de análisis. Puede que sea necesario calibrar los transductores y equipos de grabación y guardar registros de calibración para la corrección de los datos registrados. Se desarrollan uno o varios prototipos. A menudo se trata de un proceso, donde el software y el hardware se desarrollan en paralelo. Con cada prototipo realizado la diferencia con el producto final se estrecha. Con cada paso, la incertidumbre sobre la validez de las conclusiones alcanzadas anteriormente se reduce aún más.

En esta etapa se aplican los siguientes niveles de prueba:

- Prueba unitaria y de integración de software.
- Prueba de integración de software.
- Prueba de integración del hardware/software.
- Prueba de integración de los sistemas.
- Prueba del entorno.

Para las pruebas unitarias de software (SW/U) y de integración del software (SW/I), se crea un banco de pruebas que es comparable a la del entorno de prueba para el modelo de simulación. La diferencia radica en el objetivo que se ejecuta. En la fase del prototipo del objetivo de prueba es una versión ejecutable de una unidad de software, o un conjunto de unidades de software integrado, que se desarrolla sobre la base del diseño o generado a partir del modelo de simulación.

En la prueba de integración hardware/software (HW/SW/I) el objetivo de pruebas es la parte del hardware en el que se carga el software embebido. El software está incorporado en el hardware en la memoria, por lo general EEPROM. El hardware puede tener una configuración experimental, por ejemplo, una placa de circuito con

cableado que contiene varios componentes, incluyendo la memoria. El término “experimental” indica que el hardware utilizado no se seguirá desarrollando. El objetivo de estas pruebas es comprobar la correcta ejecución del software incluido en el procesador en cooperación con los periféricos hardware.

Todas las partes del hardware que el sistema embebido contiene están reunidas en las pruebas de integración de los sistemas, generalmente en un prototipo de placa de circuito impreso. Obviamente, todas las pruebas anteriores SW/U, SW/I, y el nivel de integración de HW/SW tendrán que ser ejecutadas con éxito. Todo el software del sistema tiene que ser cargado. El objetivo de las pruebas es la integración del sistema para comprobar el correcto funcionamiento del sistema embebido completo. El entorno de pruebas es muy similar a la de las pruebas de HW/SW/I, después de todo, el sistema completo es también una parte del hardware que contiene software. Una diferencia puede encontrarse en el hecho de que el prototipo de circuito impreso del sistema completo se proporciona con su final de E/S y conectores de alimentación. La oferta de los estímulos y el seguimiento de las señales de salida, posiblemente en combinación con la simulación dinámica, se llevará a cabo a través de estos conectores. Después de la monitorización de la señal en los conectores, las pruebas en circuito pueden tener lugar en lugares predefinidos en la placa de circuito impreso.

El objetivo de las pruebas de entorno en esta etapa es detectar y corregir los problemas en este ámbito lo antes posible. Esto está en contraste con las pruebas de entorno durante la etapa de pre-producción, que sirven para demostrar la conformidad. Las pruebas de entorno tienen lugar preferentemente en los prototipos que han alcanzado un nivel de madurez suficiente. Estos prototipos se construyen sobre las placas de circuitos impresos y de hardware con las especificaciones correctas. Si se requieren medidas en el sistema embebido tendrán que ser tomadas en el prototipo que se somete a las pruebas.

8.1.4. Tercera etapa: pre-producción

La etapa de pre-producción consiste en la construcción de una unidad que se utiliza para proporcionar las pruebas finales de que todos los requisitos, incluyendo el medioambiental y gubernamental.

Los objetivos de las pruebas en esta etapa son demostrar finalmente que se cumplan todos los requisitos, demostrar la conformidad con las normas gubernamentales y medioambientales, demostrar que el sistema puede ser construido en un entorno de

producción dentro de fecha programada con el esfuerzo programado, demostrar que el sistema se puede mantener en un entorno de la vida real y que se cumplan los requisitos de tiempo medio de reparación y para mostrar el producto a los clientes.

La unidad de pre-producción es un sistema real probado en el entorno de la vida real. Es la culminación de todos los esfuerzos de las etapas anteriores. La unidad de pre-producción es equivalente a las unidades de producción finales. La diferencia con las unidades de producción es que las unidades de pre-producción todavía pueden tener disposiciones de pruebas como compensaciones en la recogida de la señal, etc. Sin embargo, estas disposiciones son de calidad de la producción en lugar de la calidad del laboratorio. Puede ser necesario tener más de un modelo porque las pruebas pueden funcionar en paralelo o unidades pueden ser destruidos en las pruebas.

La unidad de pre-producción se somete a las pruebas de la vida real de acuerdo con uno o más escenarios de prueba predeterminados. Otras unidades de pre-producción pueden ser sometidas a las pruebas de calificación ambiental. En las etapas anteriores, las señales de entrada pueden haber sido generadas a partir del modelo de simulación y los datos de salida pueden haber sido controlados en red o almacenados en el disco duro para su posterior análisis.

Debe prestarse la debida atención a la calidad y la calibración de los instrumentos de pruebas ya que, si no se puede confiar en ellos, las pruebas no sirven para nada.

Los tipos de pruebas que se aplican a esta etapa son:

- Pruebas de aceptación del sistema.
- Pruebas de calificación.
- Pruebas de ejecución de seguridad.
- Pruebas de laboratorios de producción y mantenimiento.
- Inspección y pruebas gubernamentales.

8.1.5. Cuarta etapa: post-producción

La creación de prototipos y la etapa de pre-producción, finalmente, dan lugar a una “liberación de la producción” del sistema. Esto significa que el sistema es de suficiente calidad para ser vendido a los clientes. Si el proceso de producción es de calidad insuficiente, a continuación, los productos serán de calidad insuficiente, a pesar de una liberación exitosa para la producción. Por lo tanto, antes del inicio de la producción a gran escala, la organización puede requerir medidas adicionales a fin de hacer de este

proceso de producción, controlar y asegurar la calidad de los productos manufacturados.

Se debe considerar el desarrollo y las pruebas de las instalaciones de producción, la inspección del primer artículo y las pruebas de producción y mantenimiento.

8.2. HERRAMIENTAS

8.2.1. Introducción

El software ahora compone la mayor parte de los sistemas embebidos y la importancia del este sigue creciendo. Este software se vuelve más complejo y se implementa en sistemas críticos muy a menudo. Esto tiene un impacto importante en las pruebas ya que son más complicadas. Con el uso adecuado de las herramientas de pruebas es posible bajar la complejidad con muy buena calidad.

Una herramienta de pruebas es un recurso automatizado que ofrece soporte a una o más de las actividades de prueba, tales como la planificación y el control, la especificación, la construcción de datos iniciales, la ejecución de las pruebas y la evaluación.

8.2.2. Clasificación

Las herramientas están disponibles para todas las fases del ciclo de vida de las pruebas. La mayoría de las herramientas están disponibles comercialmente, pero también es posible que se desarrollen específicamente. Estas herramientas se pueden clasificar en varios grupos:

- Planificación y control:
    - Herramienta de gestión de defectos.
    - Herramienta de gestión de pruebas.
    - Herramienta de gestión de configuración.
    - Herramienta de programación y seguimiento de progresos.
- Preparación:
    - Analizadores.
- Especificación:
    - Generador de casos de prueba.
- Ejecución:

    - generador de datos.
    - Herramienta de registro y reproducción.
    - Herramienta de prueba de carga y estrés.
    - Simulador.
    - Depurador.
    - Analizador de código fuente.
    - Herramienta de detección de errores.
    - Analizador de rendimiento;
    - Analizador de cobertura de código;
    - Analizador de hilos y eventos;
    - Herramienta de detección de amenazas.
- Finalización:
    - Herramienta de programación y seguimiento de progresos.

ANEXOS

ANEXO I. CASO PRÁCTICO

I.1. TRANSICIÓN DE ESTADOS

I.1.1. Descripción del caso

Se sigue el funcionamiento del sistema descrito en el apartado 5.5 (Técnica de diseño: Transición de estados). A modo de recordatorio, se trata de un ascensor de cuatro pisos. Cuando se encuentra en cualquiera de ellos, el sistema permanecerá en reposo esperando la siguiente acción a realizar. Si se pulsa cualquier botón, el ascensor acudirá a la planta indicada[2].

I.1.2. Diagrama de estados

En la Figura I.1 se muestra el diagrama de estados o transiciones que se ha utilizado para llevar a cabo el sistema.

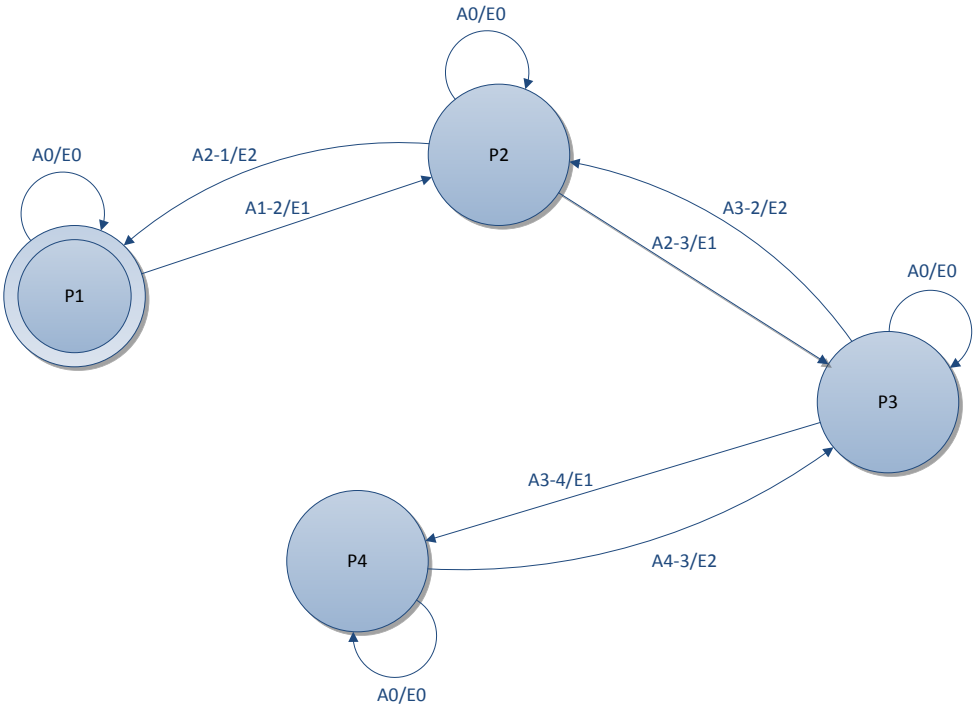


Figura I.1: Diagrama de estados del caso práctico

I.1.3. Tabla de estados y eventos

En la Tabla I.1 se muestra la tabla de estados y eventos extraída del diagrama de estados de la Figura I.1.

[2] En la simulación del sistema se usan intervalos de tiempo de 5 segundos entre planta y planta en lugar de un sensor.

Tabla I.1: Tabla de estados y eventos del caso práctico

TABLA DE ESTADOS Y EVENTOS DEL ASCENSOR				
	P1	P2	P3	P4
E1	P2	P3	P4	x
E2	x	P1	P2	P3
E0	P1	P2	P3	P4

I.1.4. Árbol de transiciones

En la Figura I.2 se muestra el árbol de transiciones extraída del diagrama de estados de la Figura I.1 y la tabla de estados eventos de la Tabla I.1.

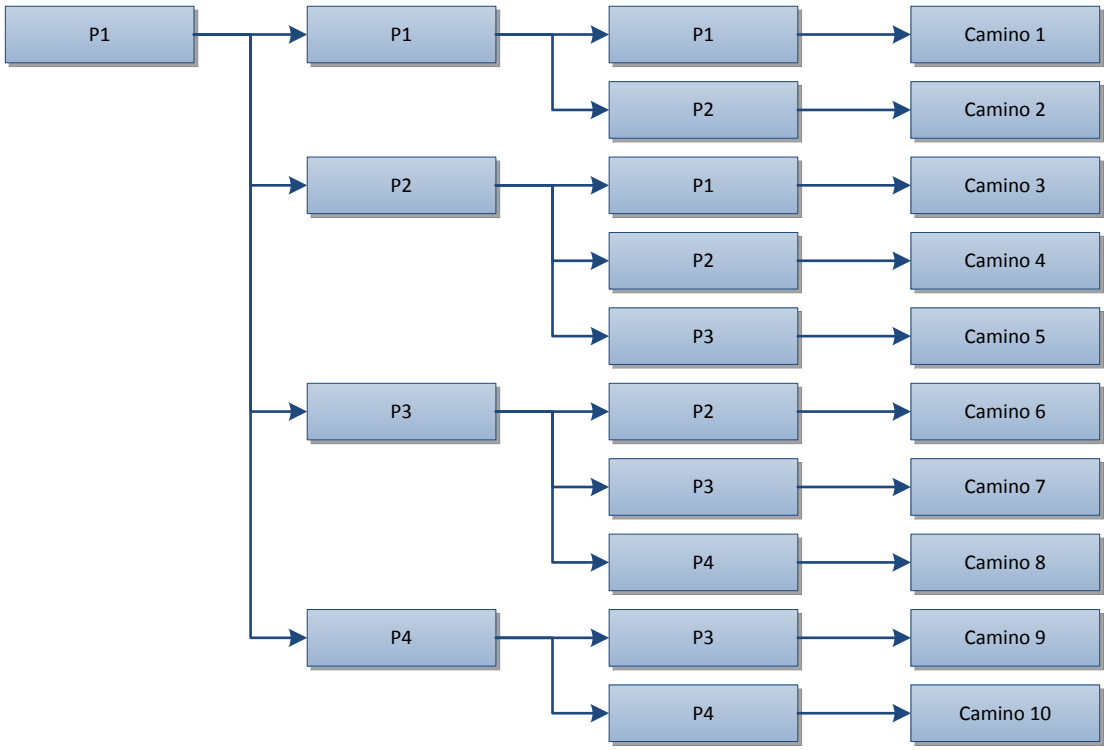


Figura I.2: Árbol de transiciones del caso práctico

I.1.5. Guion de casos de prueba legales

En la Tabla I.2 se muestra el guion de casos de prueba legales del sistema anteriormente descrito.

Tabla I.2: Guion de pruebas de los casos legales del caso práctico

GUION DE PRUEBAS DE LOS CASOS LEGALES				
ENTRADA			RESULTADO ESPERADO	
ID	Evento	Situación inicial	Acción	Estado
CL1.1 CL1.2	0 1	0 1	0 1-2	1 2
CL2.1 CL2.2 CL2.3	2 0 1	2 0 1	2-1 0 2-3	1 2 3
CL3.1 CL3.2 CL3.3	2 0 1	2 0 1	3-2 0 3-4	2 3 4
CL4.1 CL4.2	2 0	2 0	3-3 0	3 4

En la Tabla I.3 se muestra el guion de casos de prueba legales detallado del sistema anteriormente descrito.

Tabla I.3: Guion de pruebas detallado de los casos legales del caso práctico

GUION DE PRUEBAS DETALLADO DE LOS CASOS LEGALES	
CASO LEGAL	DESCRIPCIÓN
CL1.1	Estando en el piso 1, reposar con el motor del ascensor parado debido a que se ha pulsado el botón para acceder al piso 1.
CL1.2	Estando en el piso 1, subir al piso 2 accionando el motor del ascensor en modo subida si el botón pulsado es superior al piso 2.
CL2.1	Estando en el piso 2, bajar al piso 1 accionando el motor del ascensor en modo bajada si el botón pulsado es inferior al piso 2.
CL2.2	Estando en el piso 2, reposar con el motor del ascensor parado debido a que se ha pulsado el botón para acceder al piso 2.
CL2.3	Estando en el piso 2, subir al piso 3 accionando el motor del ascensor en modo subida si el botón pulsado es superior al piso 2.
CL3.1	Estando en el piso 3, bajar al piso 2 accionando el motor del ascensor en modo bajada si el botón pulsado es inferior al piso 3.
CL3.2	Estando en el piso 3, reposar con el motor del ascensor parado debido a que

CL3.3	se ha pulsado el botón para acceder al piso 3. Estando en el piso 3, subir al piso 4 accionando el motor del ascensor en modo subida si el botón pulsado es superior al piso 3.
CL4.1 CL4.2	Estando en el piso 4, bajar al piso 3 accionando el motor del ascensor en modo bajada si el botón pulsado es inferior al piso 4. Estando en el piso 4, reposar con el motor del ascensor parado debido a que se ha pulsado el botón para acceder al piso 4.

I.1.6. Guion de casos de prueba ilegales

En la Tabla I.3 se muestra el guion de casos de prueba legales del sistema anteriormente descrito.

Tabla I.4: Guion de pruebas de los casos legales del caso práctico

GUION DE PRUEBAS DE LOS CASOS ILEGALES DEL ASCENSOR				
ID	SITUACIÓN INICIAL	ESTADO	EVENTO	RESULTADO
CI1		1	2	KO
CI2	CL4.1	4	1	KO

I.1.7. Errores en la traducción de las especificaciones

Según el diagrama de estados proporcionado y realizando un estudio de los requisitos del sistema, se puede observar que existe un defecto en el diseño:

“El diagrama de estados no representa una traducción correcta de las especificaciones funcionales del sistema”.

Se puede observar que no existe definición alguna en el diagrama de estados sobre qué haría el ascensor, por ejemplo, si se encuentra en el piso 1 y se quiere subir al piso 4. Y al contrario, si el ascensor se encuentra en el piso 4 y se quiere bajar al piso 1.

Con esta técnica y gracias a los requisitos funcionales del sistema se ha podido detectar este error grave en el sistema embebido.

I.2. ESQUEMA ELÉCTRICO

I.2.1. Esquema eléctrico utilizando Proteus

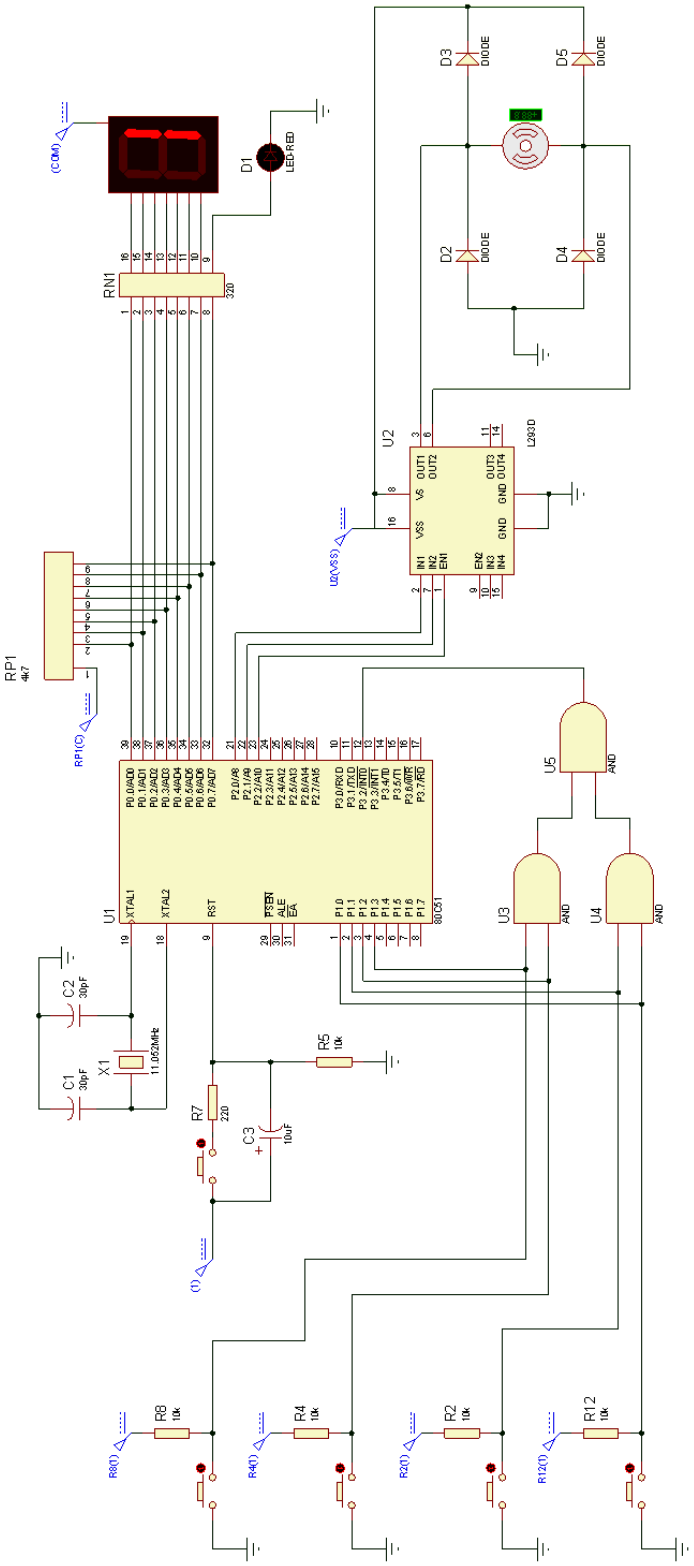


Figura I.3: Esquema eléctrico del caso práctico

I.3. SOFTWARE

I.3.1. Software del sistema utilizando uVision Keil

```
#include <reg52.h>

#define RETARDO 450   // Retardo de un segundo con un xtal = 11.0592 MHz

unsigned char code segmentos[] = {0x00, 0x79, 0x24, 0x30, 0x19};
unsigned char code pulsador[] = {0x00, 0x01, 0x02, 0x00, 0x03, 0x00, 0x00, 0x00, 0x04};
unsigned char estado = 1;
unsigned char piso = 1;

void inicializacion(void);    // Inicializa el microcontrolador
void delay(unsigned char s);  // Retardo de 3 segundos


void main(void)
{
    inicializacion();

    while(1)
    {
        switch (estado)
        {
            case 1: // Ascensor en la primera planta (Piso 1)
            {
                if(piso != estado)
                {
                    if(piso > estado)        // Subir
                    {
                        P2 = 0x06;           // Modo subida del ascensor
                        delay(5);            // Espera 5 segundos

                        estado = 2;
                        P0 = segmentos[estado];

                        if(piso == estado)   // Parar
                        {
                            P2 = 0x00;       // Modo parada del ascensor
                            delay(2);        // Espera 2 segundos
                        }
                    }
                }
            }
            break;

            case 2: // Ascensor en la segunda planta (Piso 2)
            {
                if(piso != estado)
                {
                    if(piso < estado)        // Subir
                    {
                        P2 = 0x06;           // Modo subida del ascensor
                        delay(5);            // Espera 5 segundos

                        estado = 3;
                        P0 = segmentos[estado];

                        if(piso == estado)   // Parar
                        {
                            P2 = 0x00;       // Modo parada del ascensor
                            delay(2);        // Espera 2 segundos
                        }
                    }
                    else                     // Bajar
                    {
                        P2 = 0x05;           // Modo bajada del ascensor
                        delay(5);            // Espera 5 segundos

                        estado = 1;
                        P0 = segmentos[estado];

                        if(piso == estado)   // Parar
                        {
```

```
                            P2 = 0x00;      // Modo parada del ascensor
                            delay(2);       // Espera 2 segundos
                        }
                    }
                }
            }
            break;

            case 3: // Ascensor en la tercera planta (Piso 3)
            {
                if(piso != estado)
                {
                    if(piso > estado)       // Subir
                    {
                        P2 = 0x06;          // Modo subida del ascensor
                        delay(5);           // Espera 5 segundos

                        estado = 4;
                        P0 = segmentos[estado];

                        if(piso == estado)  // Parar
                        {
                            P2 = 0x00;      // Modo parada del ascensor
                            delay(2);       // Espera 2 segundos
                        }
                    }
                    else                    // Bajar
                    {
                        P2 = 0x05;          // Modo bajada del ascensor
                        delay(5);           // Espera 5 segundos

                        estado = 2;
                        P0 = segmentos[estado];

                        if(piso == estado)  // Parar
                        {
                            P2 = 0x00;      // Modo parada del ascensor
                            delay(2);       // Espera 2 segundos
                        }
                    }
                }
            }
            break;

            case 4: // Ascensor en la cuarta planta (Piso 4)
            {
                if(piso != estado)
                {
                    if(piso < estado)       // Bajar
                    {
                        P2 = 0x05;          // Modo bajada del ascensor
                        delay(5);           // Espera 5 segundos

                        estado = 3;
                        P0 = segmentos[estado];

                        if(piso == estado)  // Parar
                        {
                            P2 = 0x00;      // Modo parada del ascensor
                            delay(2);       // Espera 2 segundos
                        }
                    }
                }
            }
            break;

            default:
            break;
        }
    }
}

void inicializacion(void)
{
    P0 = segmentos[1];

    IE = 0x89;      // EA=1, ET1=1, EX0=1 (Timer 1 e interrupción externa 0)
```

```
    TCON = 0x01;     // IT0 = 1 (Interrupción externa 0 activa por flanco de bajada)
}

void delay(unsigned char s)
{
    unsigned int i, j, k;

    for(i=0; i<s; i++)
        for(j=0; j<RETARDO; j++)
            for(k=0; k<255; k++);
}

void interrupcionExterna0(void) interrupt 0
{
    if(piso == estado)
        piso = pulsador[255 - P1];
}
```

I.3.2. Errores en la traducción en código

Si se sigue los casos de prueba diseñados, al ejecutar el siguiente caso se detectará un funcionamiento extraño en el sistema:

Tabla I.5: Guion de pruebas con errores detectados del caso práctico

GUION DE PRUEBAS DETALLADO DE LOS CASOS LEGALES	
CASO LEGAL	DESCRIPCIÓN
CL1.1	Estando en el piso 1, reposar con el motor del ascensor parado debido a que se ha pulsado el botón para acceder al piso 1.
CL1.2	Estando en el piso 1, subir al piso 2 accionando el motor del ascensor en modo subida si el botón pulsado es superior al piso 2.
CL2.1	Estando en el piso 2, bajar al piso 1 accionando el motor del ascensor en modo bajada si el botón pulsado es inferior al piso 2.
CL2.2	Estando en el piso 2, reposar con el motor del ascensor parado debido a que se ha pulsado el botón para acceder al piso 2.
CL2.3	Estando en el piso 2, subir al piso 3 accionando el motor del ascensor en modo subida si el botón pulsado es superior al piso 2.
CL3.1	Estando en el piso 3, bajar al piso 2 accionando el motor del ascensor en modo bajada si el botón pulsado es inferior al piso 3.
CL3.2	Estando en el piso 3, reposar con el motor del ascensor parado debido a que se ha pulsado el botón para acceder al piso 3.
CL3.3	Estando en el piso 3, subir al piso 4 accionando el motor del ascensor en modo subida si el botón pulsado es superior al piso 3.
CL4.1	Estando en el piso 4, bajar al piso 3 accionando el motor del ascensor en modo bajada si el botón pulsado es inferior al piso 4.
CL4.2	Estando en el piso 4, reposar con el motor del ascensor parado debido a que se ha pulsado el botón para acceder al piso 4.

La causa del defecto es una mala traducción del diagrama de estados del ascensor en código:

"Mala traducción de los diagramas de estados en código. Cada vez es más común que esta traducción se realice automáticamente. Si es así, el código generado es una representación exacta del comportamiento de los diagramas de estados. El uso de los diagramas de estados como la base para el diseño de las pruebas en este caso es, por lo tanto, poco útil y la aplicación de la técnica de pruebas basada en el estado es redundante. Sin embargo, si la codificación se realiza sin el uso de herramientas, a continuación, se debe utilizar la técnica de pruebas basada en el estado".

Gracias a la definición de los casos de prueba que han fallado, se puede acceder fácilmente a la parte del código defectuosa:

```
case 2: // Ascensor en la segunda planta (Piso 2)
            {
                if(piso != estado)
                {
                    if(piso < estado)        // Subir
                    {
                        P2 = 0x06;           // Modo subida del ascensor
                        delay();

                        estado = 3;
                        P0 = segmentos[estado];

                        if(piso == estado)
                        {
                            P2 = 0x00;
                            delay();
                        }
                    }
                    else                     // Bajar
                    {
                        P2 = 0x05;           // Modo bajada del ascensor
                        delay();

                        estado = 1;
                        P0 = segmentos[estado];

                        if(piso == estado)
                        {
                            P2 = 0x00;
                            delay();
                        }
                    }
                }
            }
            break;
```

La condición que se muestra en las expresiones marcadas en color "rojo" es errónea, ya que si se quiere subir al segundo piso, el ascensor reacciona activando el motor en modo bajada, y si se quiere bajar al segundo piso, al contrario que en el caso anterior, el ascensor accionará el motor en modo subida.

Definiciones

Banco de pruebas:

Software/hardware que ofrece estímulos para un objetivo y registra los resultados de la misma.

Base de pruebas:

Toda la documentación del sistema que se utiliza como base para el diseño de los casos de prueba.

Caja Blanca:

Propiedades internas de un objeto para obtener los casos de prueba, utilizando el conocimiento de su estructura interna.

Caja negra:

Propiedades externas de un objeto para obtener los casos de prueba, sin utilizar el conocimiento de su estructura interna.

Calidad:

Propiedades externas de un objeto para obtener los casos de prueba, sin utilizar el conocimiento de su estructura interna.

Caso de prueba:

Conjunto de entradas, condiciones previas y los resultados esperados diseñados para un objetivo en particular, como realizar un camino de un programa en particular o verificar el cumplimiento de un requerimiento específico.

Ciclo de vida:

Estructura de un proceso mediante su división en fases y descripción de lo que se deben realizar y en qué orden las actividades.

Clase de equivalencia:

Porción de los dominios de entrada o salida del componente para que el componente sea el mismo que el descrito en su la especificación.

Cobertura:

Relación entre lo que se prueba y lo que puede ser probado, que se expresa como un porcentaje.

Escenario de pruebas:

Plan de pruebas que coordina la ejecución de varios guiones de prueba.

Especificación:

Documento que describe en detalle las características de un producto con vistas a su funcionalidad prevista.

Guion de pruebas:

Descripción de cómo se hacen las pruebas. Contiene las acciones relacionadas con los casos de prueba e indica la secuencia de ejecución.

Nivel de prueba:

Grupo de actividades que se organizan y gestiona juntas, se puede hacer una división en las pruebas de alto y bajo nivel.

Objetivo de las pruebas:

Objetivos que se requieren para probar el sistema o parte de él.

Plan de pruebas:

Plan del proyecto que contiene toda la información esencial para la gestión de un proyecto de pruebas.

Proceso de pruebas:

Colección de herramientas, técnicas y métodos de trabajo que se utilizan para realizar las pruebas del sistema.

Requisito funcional:

Documento que describe en detalle las características de un producto con vistas a su funcionalidad prevista.

Sistema embebido:

Sistema que interactúa con el mundo físico real utilizando actores y sensores.

Situación inicial:

Estado en el que un sistema debe estar al inicio de un caso de prueba.

Técnica de diseño:

Método estandarizado para obtener los casos de prueba a partir de una base de pruebas.

TEmb:

Método para probar el software incorporado al sistema.

Tipo de prueba:

Grupo de actividades dirigidas a la comprobación de un sistema con un número de características de calidad relacionadas.

GLOSARIO

S

T

Bibliografía

Allan, R Hambley. 2001. *Electrónica.* s.l. : Prentice Hall, 2001.

Angulo, J M. 1996. *Electrónica Digital Moderna.* s.l. : Paraninfo, 1996.

Beizer, B. 1990. *Software Testing Techniques.* s.l. : International Thomson Computer Press, 1990.

Binder, R V. 2000. *Testing Object-oriented Systems: Models, Patterns and Tools.* s.l. : Addison-Wesley, 2000.

Broekman, Bart y Notenboom, Edwin . 2003. *Testing Embedded Software.* s.l. : Addison-Wesley, 2003.

Ceballos, Fco. Javier. 2005. *Curso de programación C/C++.* s.l. : RA-MA, 2005.

Graham, Dorothy , y otros. 2012. *Foundations of software testing: Istqb Certification.* s.l. : Cengage Learning EMEA, 2012.

Koomen, Tim, y otros. 2006. *TMap® Next for result-driven testing.* s.l. : UTN Publishers, 's-Hertogenbosch, 2006.